INGENIERÍA INDUSTRIAL
AUTOMÁTICA Y ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA

"INTERFACES DE COMUNICACIÓN PARA EL SOFTWARE CPN TOOLS"

RESUMEN EN ESPAÑOL

Autor: Carlos Arias Fernández

Tutor: Luiz Edival de Souza

Institución:

UNIFEI (Universidad Federal de Itajubá – Brasil)

Coordinador académico: **Elisa María Ruiz Navas**
Cotutor: **Esther Guerra Sánchez**

Fecha de lectura: 1/09/2010
Tribunal:

Guilherme Sousa Bastos
Kleber Roberto da Silva Santos

El presente Proyecto Final de Carrera fue realizado en el Centro de Referencia de las Tecnologías de Información (CRTi) de la Universidad Federal de Itajubá (UNIFEI) en Brasil.

El proyecto consiste en el desarrollo de un interfaz que comunique el software CPN Tools con el exterior mediante el protocolo TCP/IP. A continuación se da una breve explicación sobre las redes de Petri coloreadas (CPN), el software CPN Tools y la necesidad de desarrollar este proyecto.

Palabras clave: redes de Petri, CPN, CPN Tools, interfaz, protocolo, TCP/IP.

INTRODUCCIÓN

Desde la aparición de las redes de Petri, el modelado de sistemas distribuidos discretos tomó forma y se ha venido usando como herramienta para modelar, analizar y evaluar el comportamiento de estos desde entonces.

Una red de Petri consta de 4 elementos: estados (representados por elipses), transiciones (por rectángulos o líneas en algunos casos), arcos (que unen los arcos a las transiciones y viceversa) y los *tokens* o marcas que indican en que estados se encuentra el sistema.

En una posterior evolución de las redes de Petri, aparecieron las redes de Petri coloreadas, desde ahora CPN, por sus siglas en inglés (Coloured Petri Nets), donde fueron añadidos distintos tipos de datos atribuidos a los *tokens*. El término "coloreadas" en estas redes de Petri, viene por la existencia de varios "colores" de datos en los *tokens*, lo que refleja la existencia de varios tipos de datos y de distinta complejidad cada uno. Los *tokens* van asociados a un color de datos, o *colour set*, en donde está definida la estructura del mismo y los componentes. Estos datos, aparte de poder ser de distinta tipo, pueden ser complejos y personalizables, de una manera análoga a los objetos de los lenguajes de programación orientada a objetos. Aparte, las CPN combinan las características de las redes de Petri, con las de un lenguaje de programación, ya que se permiten editar las expresiones de los arcos y las transiciones, así como introducir funciones. Las CPN pueden además tener estructura jerárquica, es decir, que pueden construirse subredes que representan un sólo estado o una transición en la CPN superior.

Por último, también pueden considerar la variable temporal, estableciendo tiempos determinados o basados en una distribución estadística para el disparo de una transición.

El software con el que se ha trabajado en este proyecto, CPN Tools, es una aplicación que permite la edición, simulación y el análisis de las CPN. Este software fue desarrollado en SML (Standard Meta Language) en el departamento de informática de la Universidad de Aarhus en Dinamarca y se distribuye bajo licencia gratuita.

Desde el CRTi (Centro de Referencia de las Tecnologías de la Información) de la UNIFEI, se lleva trabajando y divulgando el uso de las CPN para ambientes de manufactura entre otros. Gracias a las CPN, se pueden identificar previamente puntos de estrangulamiento, calcular rendimientos de fabricación de piezas y calcular la probabilidad de que un error ocurra y otros parámetros, sin tener que recurrir a la construcción de una instalación o prototipo.

La intención preliminar del CRTi es utilizar las CPN para gerenciar el funcionamiento de un sistema en tiempo real, para lo cual se utilizará CPN Tools en modo simulación de manera que sean los elementos externos de este sistema (por ejemplo un sensor), los que manden una información para que provoque el disparo de una transición. Para realizar esta comunicación tendrá que usarse la biblioteca de comunicaciones de CPN Tools.

La biblioteca de comunicaciones de CPN Tools, fue presentada en un congreso sobre CPN en 2001 y permitía por primera vez la comunicación a través del protocolo TCP/IP con el exterior del aplicativo. Está biblioteca permite establecer conexiones múltiples y simultáneas, transmitir datos polifórmicos de acuerdo con la estructura definida en algún *colour set* ya definido.

La arquitectura de esta biblioteca se compone de tres capas:

- Capa de comunicaciones (Communications Layer): Define las funciones básicas para realizar la comunicación a través de sockets TCP/IP: conexión como cliente, como servidor, envío, recepción y cierre de la conexión.
- Capa de mensajes (Messaging Layer): Basada en la anterior, proporciona una abstracción de envío de paquetes de datos. Para ello fragmenta la información enviada en bloques de 128 bytes utilizando el primero de ellos como indicador

tanto de la longitud del paquete como de si se esperan más paquetes que componen el dato enviado.

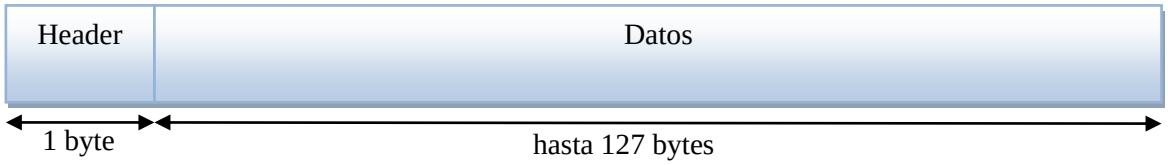


Figura 1: Distribución de un mensaje en CPN Tools

- Capa de gestión de la comunicación (Communication Management Layer): Basada en las anteriores, añade la posibilidad de identificar la conexión, cuando se utilizan varias conexiones al mismo tiempo. Además, necesita como argumento una función que codifique (en el caso del envío) o decodifique (recepción) el tipo de dato requerido (ajustándose al definido en el *colour set*).

OBJETIVOS

Desde el CRTi, se plantearon los siguientes objetivos:

- Comunicar de forma simplificada el software CPN Tools con el exterior
  - Análisis de viabilidad con los lenguajes Java, C y C#
  - Análisis del software CPN Tools en relación a sus comunicaciones.
  - Desarrollar una aplicación en el lenguaje escogido (preferentemente en C#) que actúe como interfaz de comunicación entre CPN Tools y el ambiente externo.

ANÁLISIS DE VIABILIDAD

Se escogió C# como lenguaje a utilizar ya que es un lenguaje orientado a objetos (respecto a C) y era el utilizado y preferido en el CRTi (respecto a Java)

LIMITACIONES DE LAS COMUNICACIONES

Durante el análisis del software CPN Tools, se descubrió algúna deficiencia en las comunicaciones. A pesar de que la biblioteca de comunicaciones de CPN Tools

permite múltiples conexiones y simultaneas, el posterior estudio de la dinámica de éstas derivó en las siguientes limitaciones.

Sincronismo: Ejecutando las instrucciones de recepción y de abrir una conexión como servidor, el programa queda bloqueado hasta que se recibe un dato o un cliente se conecta.

Ejecución puntual: Estas mismas instrucciones sólo se ejecutan una vez y es necesario ejecutarlas de nuevo si quieren ser usadas nuevamente. No existe la posibilidad por ejemplo de dejar un servidor abierto de forma continua. Si de todas formas fuera posible, debido a la limitación anterior, el sistema quedaría bloqueado indefinidamente.

Identificación local de las conexiones: Aunque las conexiones poseen un identificador, este sólo es accesible y reconocible dentro de la CPN donde la conexión fue establecida. Por tanto, una misma conexión será identificada de forma distinta desde la red que abrió el servidor y la que se conectó a él como cliente.

DESARROLLO DE LA INTERFAZ

Para resolver los problemas que suponen las limitaciones de la conexión anteriormente planteados, fueron establecidos los siguientes requisitos:

- Múltiples conexiones: Posibilitando simultáneamente trabajar como cliente en varios puertos y abriendo varios servidores cada uno de ellos, aceptando varias conexiones en el mismo puerto.
- Comportamiento asíncrono: La interfaz debe comportarse de forma asíncrona para no bloquear ningún proceso simultáneo.
- Identificando todas las conexiones globalmente con un identificador numérico y *string* identificativo para mejor diferenciación para el usuario.
- Desarrollo de un protocolo que complemente al existente que incluya información del origen y el destino así como otras informaciones.

Además de estas posibilidades, fueron incluidas otras, que facilitan el posterior uso de la aplicación:

- Habilitado un modo *debug* o paso a paso en el protocolo desarrollado para poder acompañar las transacciones cuando estas estén en fase de desarrollo. También se da la opción de no utilizar el protocolo desarrollado, y utilizar simplemente el definido en la biblioteca de comunicaciones de CPN Tools anteriormente descrito.
- Inclusión automática de las funciones en SML y parámetros necesarios para el correcto funcionamiento del protocolo en la CPN.
- Monitoramiento de la simulación de la CPN remotamente, sin necesidad de tener instalado CPN Tools. Además se da la opción de preparar la CPN para su monitorización de manera automática desde la interfaz.
- Edición de las instrucciones de comunicación de una CPN dentro de la interfaz desarrollada sin obligatoriedad de escribir código en SML.

PROTOCOLO DESARROLLADO

Para poder incluir información adicional, se adicionaron cabeceras de manera compatible con los bloques de 128 bytes de CPN Tools. Estas cabeceras fueron incluidas sólo en el primer de los bloques enviados en el caso de que el paquete sea mayor de 128 bytes; los siguientes bloques que correspondan al mismo dato seguirán el protocolo de CPN Tools. Fueron creadas dos cabeceras: una de control del protocolo (Control) que ocupa 1 byte, y otra que contiene la información del origen y el destino de la transacción (Src&Dst) que ocupa entre 1 o 2 bytes. Como puede apreciarse en la Figura 2, para conservar la estructura del protocolo de CPN Tools, con sus bloques de 128 bytes, la longitud del cuerpo de datos se verá reducida hasta en 124 bytes.

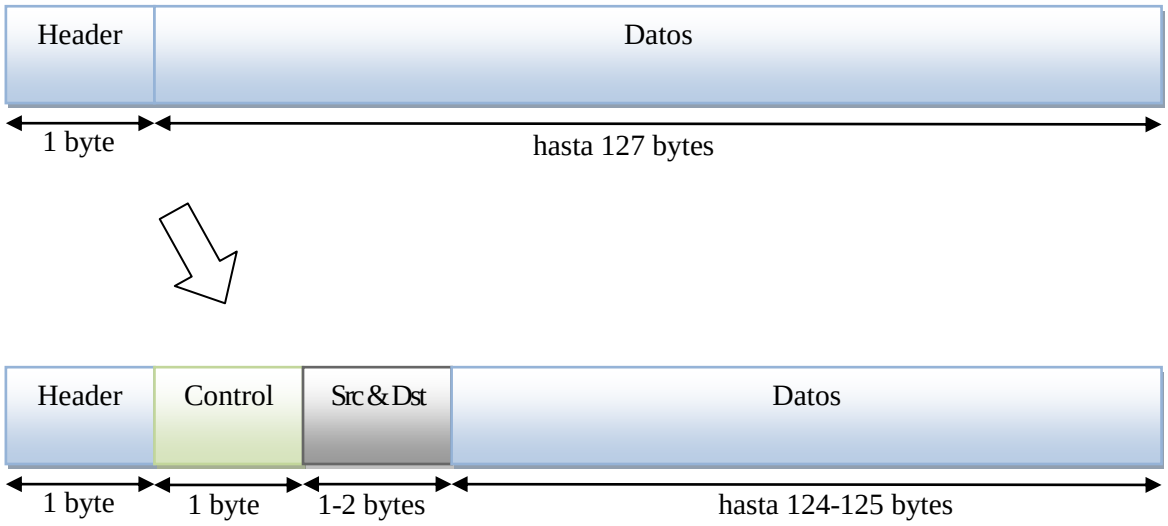


Figura 2: Distribución OSI del protocolo desarrollado

El byte de control tiene las siguientes funciones, identificadas en cada uno de sus bits. Como puede verse en la Figura 3, por el momento fueron reservadas 4 funciones, dejando 4 libres para usar en el futuro.

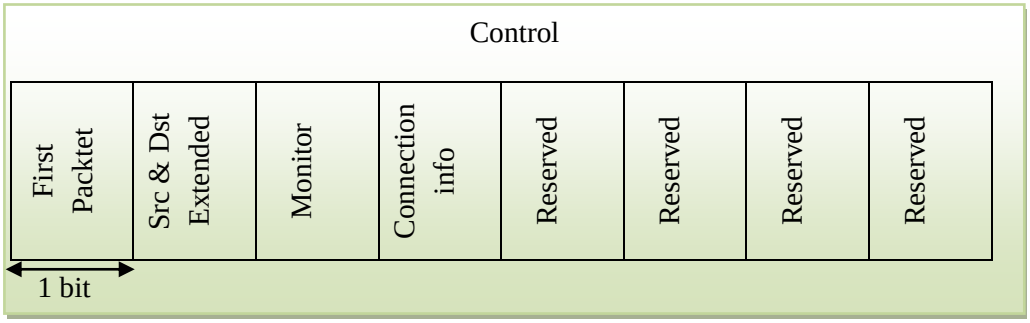


Figura 3: Distribución OSI del byte de Control

- *First Packet*: Este bit habilita el protocolo desarrollado, además sólo estará presente en el primero de los bloques de un mismo dato, utilizando la configuración normal de CPN Tools para los siguientes. Al habilitar el protocolo, es un indicador para la aplicación receptora de este dato de que el byte de Control y el *Src&Ext* están habilitados y debe ser recogida información de ellos. Es importante que sea este bit el que active el protocolo, ya que así es difícil que se active por accidente porque en general los datos transmitidos son codificados

en ASCII y ninguno supera el valor 127, con lo que consecuentemente no será activado.

- *Src & Dst Extended*: Este bit indica si el modo extendido del byte *Src&Dst* está activo o no. En el caso de que esté habilitado, y por tanto, se active el modo extendido, el protocolo permitirá identificar hasta 256 elementos conectados diferentes. En caso contrario, solamente 16.
- *Monitor*: Este bit habilitado indica que la información es para monitorización por parte de la interfaz. Este dato por tanto no tiene que ser enviado a ningún destino.
- *Connection Info:* Este bit habilitado indica que los datos recibidos contienen información de una conexión recién establecida, indicando en ellos el número de elemento conectado y su *string* identificador.

El byte *Src&Dst* (del inglés *Source and Destiny*) contiene información sobre el origen y el destino del paquete transmitido. Según si está habilitado el bit *Src&Dst Extended* del byte de Control, el byte *Src&Dst* ocupará 2 bytes o solamente uno. A continuación en la Figura 4 pueden verse las configuraciones en formato reducido y extendido.

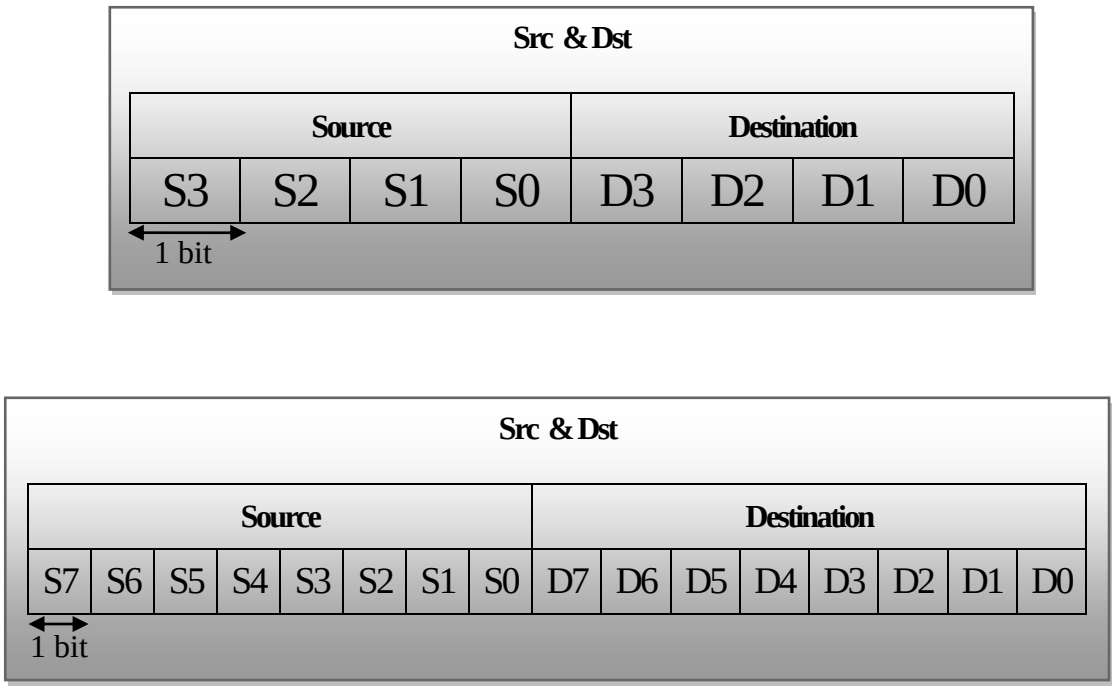


Figura 4: Configuración reducida y extendida del byte Src&Dst

DINÁMICA DEL PROTOCOLO

Para el correcto funcionamiento del protocolo, cada elemento que establezca una conexión, debe inmediatamente enviar al interfaz información que identifique a este elemento: número y *string* identificadores. En la Figura 5 se muestra detalladamente la estructura que tiene que tener esta información de la conexión. Dentro del byte de Control, están habilitados el bit que activa el protocolo así como el de *Connection Info.* El bit *Src&Dst Extended* está marcado con una E ya que dependerá del contenido del byte *Src&Dst* para estar habilitado o no. Dentro del byte *Src&Dst*, la primera mitad contiene el número identificador de la conexión. Si esta fuera mayor o igual que 16, como serán necesarios más de 4 bits para representar este número, automáticamente debe ser habilitado el modo extendido. Por otro lado, la segunda mitad muestra la longitud del *string* identificador. Si esta fuera mayor o igual a 16 caracteres, será habilitado el modo extendido igualmente. Por motivos de espacio, este *string* no podrá tener más de 124 caracteres.

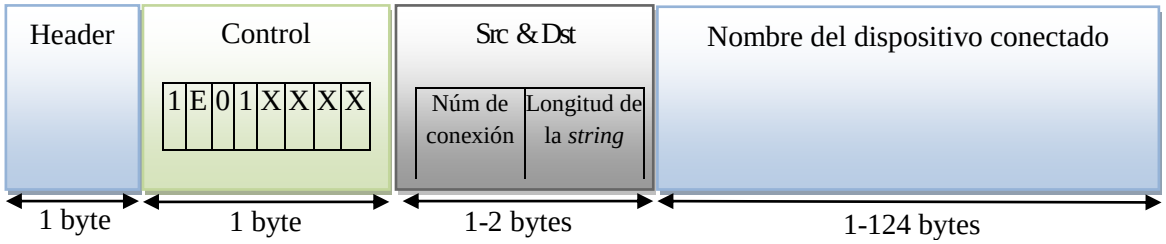


Figura 5: Configuración del envio de información de la conexión

Una vez recibida esta información, la interfaz aceptará la conexión si no hubiera otra con el mismo número de conexión. En el caso que estuviera ya usada, continuará operativa la primera conexión que utilizó ese número, y la otra quedará conectada aunque el protocolo no la considerará.

APLICACIÓN DESARROLLADA

Además de las capacidades de conexión ya comentadas, la aplicación posee las siguientes características:

- Elección entre el protocolo de CPN Tools (RAW), el desarrollado en modo automático (Enabled) y en modo *debug*.

- Visualización de una lista de las conexiones activas validadas (Ver Fig. 7)
- Monitorización de las conexiones.
- Monitorización de la simulación de una CPN conectada remotamente.
- Inclusión de la biblioteca de comunicaciones y de los parámetros de conexión de la CPN. (Ver Fig. 8)
- Preparación de la CPN para la monitorización de la simulación. (Ver Fig. 8)
- Edición de instrucciones de comunicación en la CPN (Ver Fig. 9)
- Instrucciones de comunicación contenidas en un fichero XML editable.

Esta última característica permite una cierta adaptabilidad ya que lee las instrucciones de un archivo XML creado con la aplicación donde se pueden añadir funciones de codificación y decodificación personalizables. Esto permite incorporar funciones adaptadas a las necesidades requeridas sin tener que editar el código fuente de la aplicación. El proyecto incluye un manual de usuario donde figura como crear una función de codificación o decodificación.

En la Figura 6 puede observarse el aspecto inicial del software desarrollado. Por un lado contiene campos para introducir los parámetros y poder conectarse como servidor local, remoto o abrir un servidor. Además, tiene la opción de enviar strings desde la propia interfaz, seleccionando el destino y simulando un origen. También existe un indicador con el protocolo escogido. Haciendo *click* en "Connections" aparecerá una lista de de las conexiones establecidas como se puede ver en la Figura 7. Para visualizar una CPN hay que seleccionar el archivo .cpn a leer y clicar en "Visualize" (Ver Figura 8).

ASPECTO VISUAL

- Ventana inicial

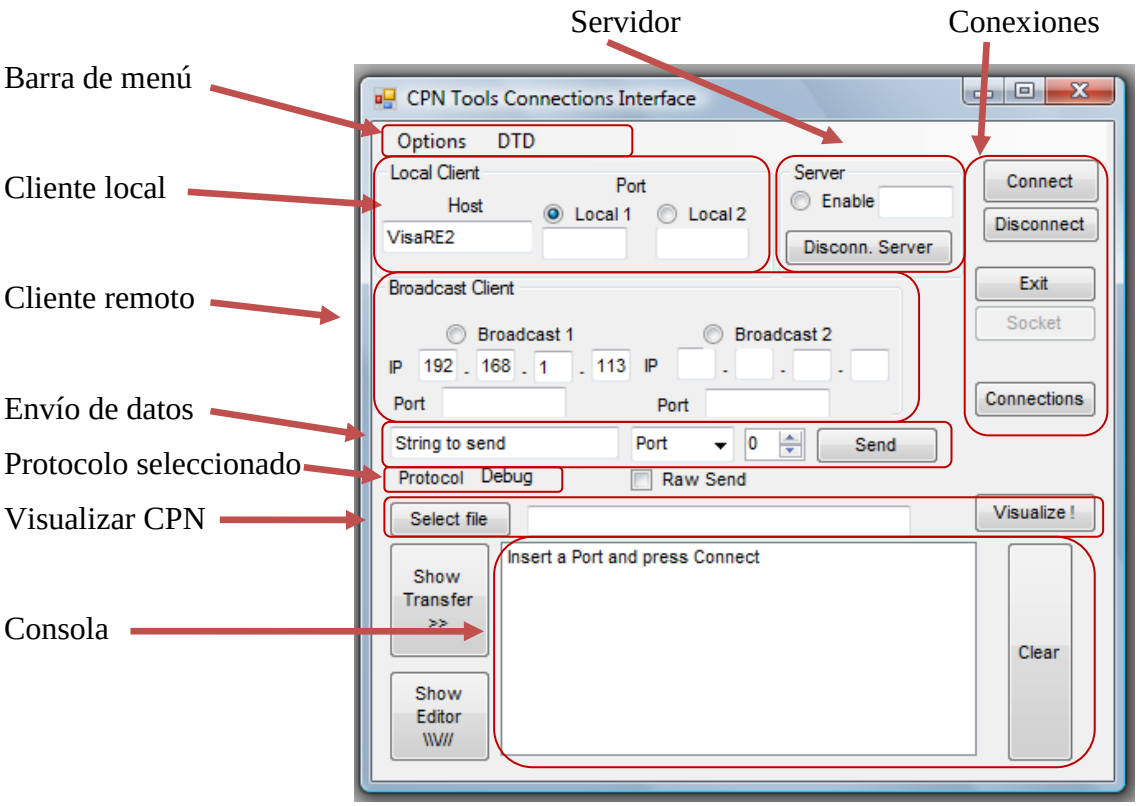


Figura 6: Aspecto inicial de la aplicación.

- Lista de conexiones

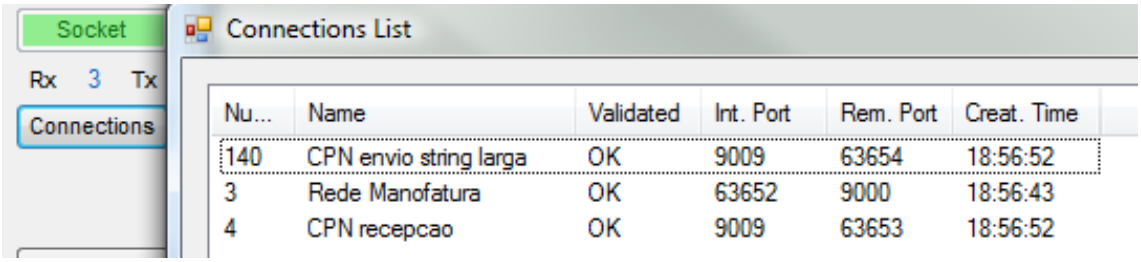


Figura 7: Lista de conexiones y monitorización

- Monitorización de la CPN y ventana de edición de parámetros y de preparación para la monitorización.

Haciendo *click* en "Monitorize Net" se abre la ventana "Monitor Options" que aparece en el lado izquierdo de la Figura 8. Dentro de ella y en la parte izquierda aparecen las opciones para preparar la red para monitorarla. En la parte derecha aparecen los parámetros de conexión, los cuales pueden ser editados desde aquí.

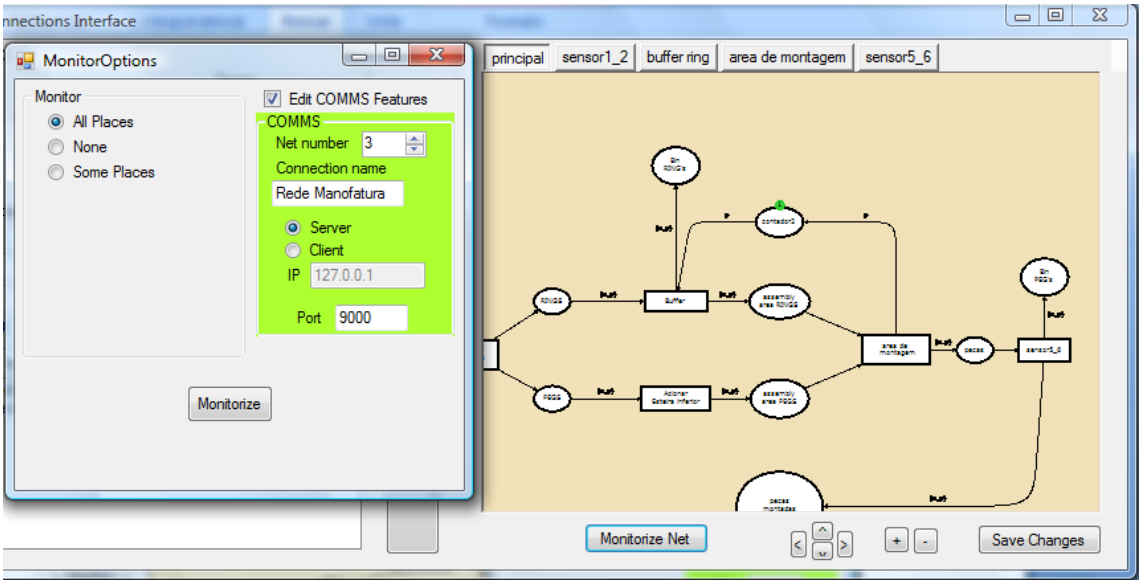


Figura 8: Monitorización de la CPN

- Edición de instrucciones de comunicación

Clicando con el botón derecho en una transición aparecen sus instrucciones de comunicación. También pueden añadirse las disponibles en la parte izquierda (éstas son leídas del archivo XML anteriormente mencionado) hacia el lado derecho, apareciendo abajo el texto que será introducido como instrucción en la transición. Naturalmente este texto puede ser editado desde aquí.

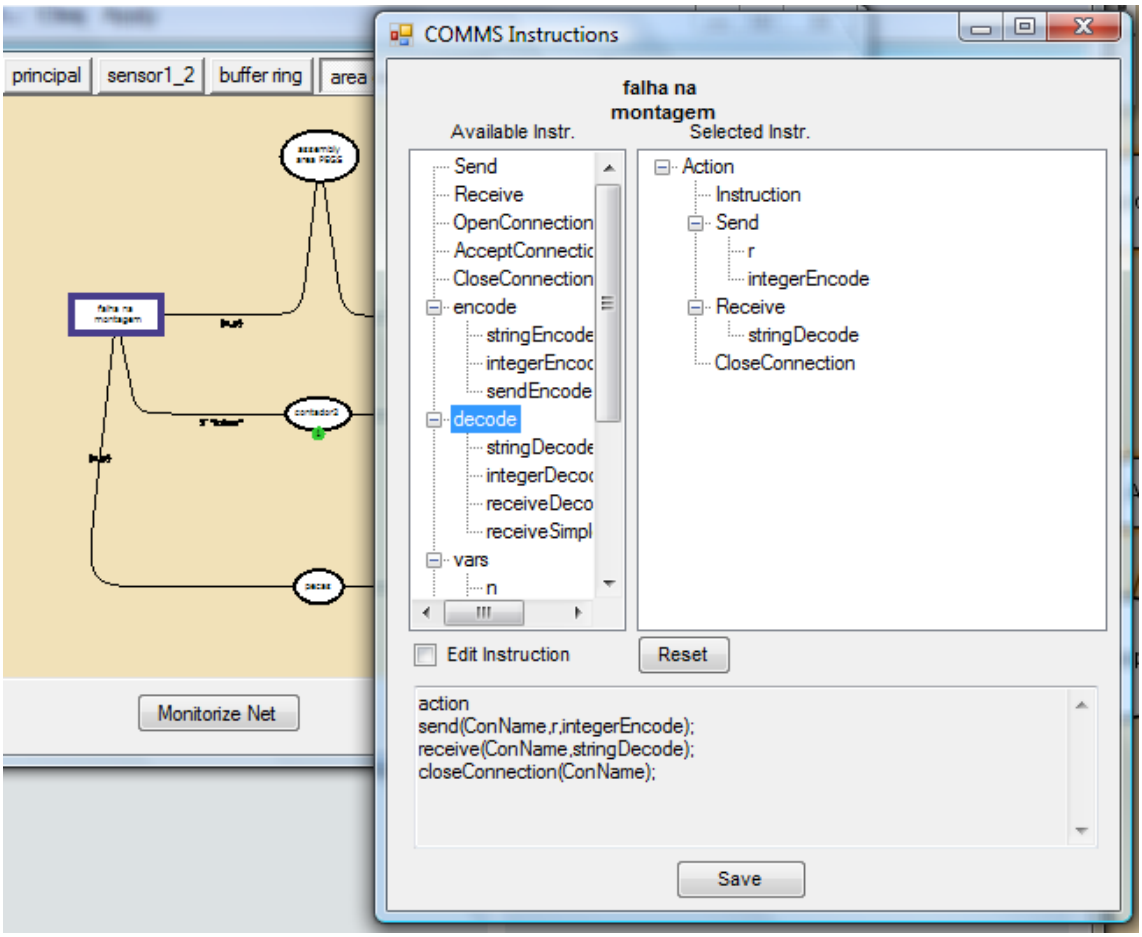


Figura 9: Instrucciones de comunicación

PRUEBAS Y ANÁLISIS DE RESULTADOS

Se realizaron continuas pruebas para la comprobación del correcto envío y recepción de todos los paquetes, ya fuera con bloques de información muy grandes (más de tres bloques de 128 bytes) o con múltiples conexiones simultáneas. Además se hicieron pruebas para determinar la velocidad de procesamiento de la información por parte de la interfaz y así valorar si era congruente el uso de la misma y su protocolo.

Prueba 1		Prueba 2	
Longitud del dato	10-16 bytes	Longitud del dato	199-201 bytes
Tiempo medio	6,48 msg	Tiempo medio	27,91 msg

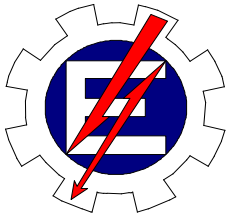
Con estos datos obtenidos, se puede considerar que no afectan substancialmente a la comunicación si los comparamos con velocidades de transferencia de archivos en redes y que además dependen de la cantidad de bytes que se transfieren. La cantidad de información a transmitir, se supone va a ser baja (menor de 128 bytes) para la aplicación en la que se pretende utilizar. No obstante, para realizar las pruebas de una manera más rigurosa debería conocerse exactamente cómo es la aplicación.

CONCLUSIONES

Este proyecto cubrió sobradamente los objetivos inicialmente planteados, ya que además de ampliar las opciones de comunicación y superar las limitaciones que CPN Tools tenía en este aspecto, fueron añadidas muchas más funcionalidades.

Gracias a la realización de este proyecto, se pueden conectar varias CPNs entre sí aparte de otros elementos a través del interfaz creado. De esta manera el siguiente paso pasaría por hacer efectiva estas conexiones y analizar los resultados. El protocolo y las demás ampliaciones incorporadas en el proyecto tienen un espíritu integrador y genérico, de manera que permite adaptarse a la cual es conectada y pueden acoger mejoras o nuevas funciones en el futuro. Ejemplos son los bits reservados en el byte de control y la opción de editar el fichero XML para incorporar nuevas funciones.

La originalidad de este proyecto, es notoria ya que no se conoce ningún interfaz parecido para conectar de manera genérica varias CPNs con el software CPN Tools de manera genérica.



UNIVERSIDADE FEDERAL DE ITAJUBÁ

IESTI - Instituto de Engenharia de Sistemas e Tecnologias da Informação

CRTi – Centro de Referência das Tecnologias da Informação

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

PROJETO FINAL DE GRADUAÇÃO

Interfaces de Comunicação para o software CPN Tools

Autor: Carlos Arias Fernández

Orientador: Prof. Dr. Luiz Edival de Souza

ITAJUBÁ

AGOSTO 2010

Dedicatória

Dedico este trabalho à minha família e a todos que me ajudaram nesta caminhada, em especial a todos aqueles que me deram inspiração ou motivação para prosseguir meus estudos de engenharia.

Agradecimentos

Quero agradecer em primeiro lugar à minha Universidade Carlos III de Madri por possibilitar-me participar do intercâmbio na UNIFEI (Universidade Federal de Itajubá) onde desenvolvi meu projeto final de graduação, além de outras disciplinas e atividades.

Agradeço ao meu orientador Prof. Dr. Luiz Edival de Souza pelas suas dicas, indicações, recomendações e correções feitas. Também aos companheiros do CRTi, pela sua disposição para resolver qualquer dúvida, especialmente ao mestrando Tamatiá Colmán, com quem conjuntamente foram resolvidas distintas questões.

Agradeço aos membros do fórum CPN Tools Support, pelas dúvidas esclarecidas, e por estarem sempre de prontidão.

Agradeço à minha orientadora Profª. Drª. Esther Guerra Sánchez da Universidade Carlos III pela ajuda com os trâmites burocráticos e as dicas que me foram dadas.

Agradeço a meus companheiros de república pela ajuda oferecida, especialmente relacionada com questões idiomáticas.

Agradeço à ARIS (Assessoria de Relações Internacionais) da UNIFEI, na qual facilitou minha integração como aluno estrangeiro e me ajudou a resolver qualquer eventualidade que surgisse.

Resumo

O CRTi propus implementar uma interface para integrar a ferramenta CPN Tools em um ambiente industrial de manufatura e para isso foi preciso desenvolver uma interface que centralizasse e controlasse as comunicações mediante o uso protocolo TCP/IP, utilizando a linguagem de programação C#.

As CPN já possuem a capacidade de comunicar-se entre si, porém era preciso desenvolver um protocolo que possibilitasse a identificação entre os elementos que compõem a rede do sistema além de outras informações relativas à comunicação de uma maneira análoga ao funcionamento de um *fieldbus*. Cabe salientar as amplas capacidades de monitoramento das conexões, assim como, a opção de escolha de diferentes protocolos desenvolvidos para efetuar as transações.

Além de possibilitar a comunicação múltipla tanto como servidor quanto como cliente, foram acrescentadas além das características de comunicação inicialmente concebidas, outras para facilitar a preparação dos arquivos próprios das CPN para adaptá-los à comunicação. Estas características compreendem desde a inserção das bibliotecas automaticamente até o monitoramento do andamento da CPN remotamente, passando pela inclusão das instruções de comunicação na CPN diretamente desde a aplicação.

Palavras-chave: CPN, CPN Tools, TCP/IP, interface, protocolo.

Abstract

CRTi (Information Technologies Reference Centre) found useful to implement an interface to integrate CPN Tools within a manufacturing cell. In order to do that, this project consists in an interface that manages connections between CPN Tools and external nodes through TCP/IP network protocol, developed using C#.

Although CPN Tools provides a communication set of instructions which operates using TCP/IP, the possibilities were quite limited, therefore it was necessary to develop a protocol which extends the communications capabilities by adding headers with communication related information in a similar way as a *fieldbus* does. In addition, network monitoring features were incorporated, as well as the possibility of choosing any of the developed protocols to perform the transfer of data.

Setting aside that it provides multiple connections either as server or as client, additional features not conceived at the beginning were later incorporated to prepare and to adapt the CPN files for the communication. These features range from the automatic addition of the communication libraries to the CPN, to offering remote monitoring of the CPN and the edition of the communication instructions directly from the developed application.

Key words: CPN, CPN Tools, TCP/IP, interface, network protocol.

Índice

Dedicatória ..... **ii**

Agradecimentos ..... **iii**

Resumo ..... **iv**

Abstract ..... **v**

Lista de figuras ..... **ix**

Lista de tabelas ..... **xi**

1. Introdução **1**

1.1. Motivação **1**

1.2. Objetivos **2**

1.3. Estudo de viabilidade **3**

1.3.1. Análise da linguagem C **3**

1.3.2. Análise da linguagem Java **3**

1.3.3. Análise da linguagem C# **3**

1.3.4. Conclusão **3**

1.4. Estrutura do documento **4**

2. Estado da arte **5**

2.1. Redes de Petri **5**

2.2. Redes de Petri coloridas **11**

2.2.1. Definição formal **12**

2.2.2. CPN não hierárquica **13**

2.2.3. CPN hierárquica **16**

2.3. CPN Tools **17**

2.3.1. Biblioteca Comms/CPN **18**

2.3.2. Camada de comunicações **19**

2.3.3. Camada de mensagem **20**

2.3.4. Camada de gerência de comunicação 21
2.4. Comunicação mediante sockets 24
3. Desenvolvimento da Interface de comunicação 26
3.1. Requisitos de projeto 26
3.2. Implementação 29
3.2.1. Comunicação simples 29
3.2.2. Comunicação entre várias CPN Tools 34
3.2.3. Comunicação sob Protocolo implementado 34
3.2.3.1. Função connect 35
3.2.3.2. Função connectEncode 37
3.2.3.3. Função sendEncode 38
3.2.3.4. Função monitorEncode 38
3.2.3.5. Função receiveDecode 39
3.2.4. Estabelecimento da conexão 42
3.2.5. Envio de dados a CPN Tools 43
3.2.6. Recepção de dados de CPN Tools 43
3.2.7. Comunicação entre várias CPN. 44
3.3. Monitorar a simulação da CPN 45
3.3.1. Leitura do arquivo .cpn 45
3.3.2. Modificação automática do arquivo .cpn para ser monitorado 47
3.4. Comunicação entre várias aplicações 47
4. Teste de software 49
4.1. Testes das comunicações. 49
4.2. Análise dos resultados 51
5. Aplicações 52
5.1. Célula de manufatura 52
5.2. CPN conectada a uma interface gráfica 59

6. Gestão do projeto 60

6.1. Seguimento cronológico 60

6.2. Ferramentas utilizadas 62

7. Conclusões e perspectivas futuras 63

8. Referências bibliográficas 65

A. Apêndice: Manual de usuário de CPN Communications Interface 67

Sumário 67

I. Requisitos e limitações 68

II. Aspecto visual. Primeiros passos. 69

Instalação do programa 69

Aspecto visual 70

II. Comunicação 71

1. Interface como cliente, CPN Tools como servidor 71

2. Interface como servidor, CPN Tools como cliente 75

III. Monitoramento 77

IV. Inclusão da biblioteca e parâmetros de comunicação 78

V. Instruções de comunicação na CPN 80

VI. Salvar as modificações da CPN 83

VII. Tabela da rede para as provas. 84

VIII. Procedimentos incorretos previstos. 84

IX. Conectar a Interface com o exterior 85

X. Construir uma função de (de)codificação 87

XI. Arquivo SMLFunctions.xml 90

Lista de figuras

Figura 2.1 Módulos de uma RdP..... 7
Figura 2.2 Ativação de uma transição 7
Figura 2.3 Exemplo de uma RdP de um elevador 10
Figura 2.4 CPN exemplo de república..... 16
Figura 2.5 CPN hierárquica 17
Figura 2.6 Detalhe do subproceso 17
Figura 2.7 Arquitetura da biblioteca Comms/CPN..... 19
Figura 2.8 Signature da camada de comunicação em SML 19
Figura 2.9 Signature da camada de mensagem em SML..... 20
Figura 2.10 Distribuição da mensagem 20
Figura 2.11 Signature da Camada de gerência de comunicação em SML..... 21
Figura 3.1 Distribuição OSI a implementar..... 26
Figura 3.2 Distribuição do byte Header 27
Figura 3.3 Descrição do *byte* de Controle..... 27
Figura 3.4 Descrição do byte Src & Dst não estendido..... 28
Figura 3.5 Descrição do byte Src & Dst estendido 28
Figura 3.6 Comunicações simples 29
Figura 3.7 CPN para o envio de dados 30
Figura 3.8 CPN na pré-conexão 30
Figura 3.9 CPN no pré-envio..... 30
Figura 3.10 CPN na pré-desconexão..... 31
Figura 3.11 Aspecto da aplicação após receber a *string*..... 31
Figura 3.12 CPN para a recepção de dados 32
Figura 3.13 CPN na pré-recepção..... 32
Figura 3.14 Aspecto da aplicação enviando a *string*..... 32
Figura 3.15 CPN após a recepção 32
Figura 3.16 CPN com envio e recepção de dados 33
Figura 3.17 Aspecto da Interface durante o envio e a recepção de dados..... 33
Figura 3.18 CPN após receber uma *string* da interface e antes de enviar outra para a interface. 34
Figura 3.19 Função connect como servidor 36
Figura 3.20 Função connect como cliente..... 36
Figura 3.21 Codificação para a conexão..... 37
Figura 3.22 Função sendEncode 38
Figura 3.23 Função monitorEncode 39
Figura 3.24 Funções para a conversão de dados de tipo *byte*, inteiros e binários .. 39
Figura 3.25 Função receiveDecode 40
Figura 3.26 Diagrama de seqüência da conexão..... 42
Figura 3.27 Diagrama de seqüência de envio de dados à CPN 43
Figura 3.28 Diagrama de seqüência de recepção de dados desde a CPN..... 44
Figura 3.29 Diagrama de seqüência de comunicação entre duas CPN..... 44
Figura 3.30 Construção de representação gráfica a partir do arquivo da CPN..... 46

Figura 3.31 Arquitetura de um possível sistema a conectar mediante a Interface 47
Figura 5.1 CPN principal da rede de manufatura (lado esquerdo) 53
Figura 5.2 CPN principal da rede de manufatura (lado direito) 54
Figura 5.3 Subrede CPN do *buffer ring* da rede de manufatura..... 55
Figura 5.4 Descrição do *buffer ring* da rede de manufatura 55
Figura 5.5 Subrede CPN da área de montagem rede de manufatura 56
Figura 5.6 Descrição da área da rede de manufatura 56
Figura 5.7 Subrede CPN do sensor 1/2 57
Figura 5.8 Subrede CPN de um dos sensores da rede de manufatura 58
Figura 5.9 Software externo de representação gráfica da CPN da república. Estado inicial..... 59
Figura 5.10 Software externo de representação gráfica da CPN da república. Estado intermediário..... 59
Figura A.1 Verificação dos requisitos durante a instalação 69
Figura A.2 Confirmação da instalação 69
Figura A.3 Aspecto da instalação no menu de inicio. 70
Figura A.4 Aspecto inicial da aplicação. 70
Figura A.5 Ferramentas de simulação de CPN Tools. 71
Figura A.6 Conexão entre a aplicação e CPN Tools. 72
Figura A.7 Indicador do estado das conexões. 72
Figura A.8 Lista de conexões ativas. 73
Figura A.9 Caixa de saída das transferências. 73
Figura A.10 Aspecto gráfico durante o envio de dados..... 74
Figura A.11 Lista de conexões. 74
Figura A.12 Servidor aberto pela aplicação. 75
Figura A.13 Servidor aceitando conexão de entrada. 75
Figura A.14 Edição dos parâmetros da comunicação. 78
Figura A.15. Seleção dos elementos a monitorar e dos parâmetros da comunicação. 79
Figura A.16 Seleção de um *place* a monitorar. 79
Figura A.17 Seleção de uma transição para incluir uma instrução. 80
Figura A.18 Aspecto da janela de instruções de comunicações..... 81
Figura A.19 Inclusão de uma instrução. 82
Figura A.20 Inclusão de um parâmetro na instrução *Send*. 82
Figura A.21 Salvar alterações na CPN 83
Figura A.22 Distribuição OSI dos blocos de dados para se comunicarem com a Interface..... 85
Figura A.23 Distribuição OSI para se comunicar com a Interface utilizando o protocolo desenvolvido 85
Figura A.24 Distribuição OSI do bloco de dados após a conexão 86
Figura A.25 Função receiveDecode simplificada..... 87
Figura A.26 Função receiveSimpleDecode 88
Figura A.27 Função receiveSimpleDecode incorporada..... 89

Lista de tabelas

Tabela 3.1 Descrição das camadas de distintos pacotes 41
Tabela 3.2 Descrição especifica dos pacotes de conexão e monitor 41
Tabela 4.1 Detalhe dos arquivos para os testes 50
Tabela 4.2 Tempos de processamento de dados para a prova E 51
Tabela 4.3 Tempos de processamento de dados para a prova F 51
Tabela 6.1 Cronograma do projeto 61
Tabela A.1 Requisitos e limitações 68
Tabela A.2 Arquivos para teste das comunicações 84

Lista de Abreviaturas

RdP: Rede de Petri

CPN: Coloured Petri Net – Rede de Petri colorida

TCP/IP: Transmission Control Protocol/ Iternet Protocol

UNIFEI: Universidade Federal de Itajubá

CRTi: Centro de Referência das Tecnologias da Informação

MIT: Massachussets Institue of Technology

UML: Unified Media Language

post-WIMP: post- Window, Icon, Menu, Pointing Device

SML: Standard Meta Language

ASCII: American Standard Code for Information Interchange

XML: eXtended Markup Language

HTTP: HiperText Transfer Protocol

OSI: Open Systems Interconnection model

1. Introdução

1.1. Motivação

Devido ao alto potencial de uso da modelagem de sistemas dinâmicos a eventos discretos baseado em redes de Petri, a Universidade de Aahrus desenvolveu uma ferramenta, chamada CPN Tools para o projeto, análise e simulação de redes de Petri coloridas. A RdP é uma linguagem gráfica para o projeto, especificação, simulação e verificação de sistemas, dentre os quais são destacados os sistemas concorrentes pela sua complexidade. Estes possuem, devido a sua concorrência, lugares de comportamento não determinístico, ou seja, no decorrer do funcionamento do sistema podem acontecer várias situações não previsíveis. O uso das CPN é bem difundido tanto no meio industrial como no acadêmico utilizado para diversos tipos de sistemas compreendendo software, hardware, protocolos de comunicação, sistemas de workflow de processos de negócio, controle de sistemas, sistemas de indústria militar e aeronáutica entre outros.

A CPN Tools já possui instruções que tornam possível a comunicação com o protocolo TCP/IP. Porém, as instruções foram desenvolvidas de uma maneira simples. Estas foram concebidas para fornecer unicamente uma comunicação pontual e síncrona, obrigando a personalizar a aplicação conectada à CPN Tools. É necessária informação adicional da comunicação, como por exemplo, o tipo de dado enviado ou conhecer a origem e o destino. Possui unicamente informação do tamanho do pacote de dados a enviar (*message framing*) que facilita a recepção correta dos dados no protocolo TCP/IP.

Portanto, havia um grande interesse de desenvolver uma aplicação que centralizasse as comunicações que ligassem a CPN Tools ou várias CPN Tools com outros elementos exteriores. Com isso acrescentou-se a possibilidade de ligar uma CPN com vários lugares ao mesmo tempo e possibilitar que a informação chegasse ao destino desejado de maneira análoga ao comportamento de um *fieldbus*. Com essa finalidade, foi necessário definir um protocolo que complementasse ao TCP/IP já existente acrescentando maior flexibilidade de uso e capacidade de comunicar.

No CRTi há a intenção, entre outras, de usar as CPN para modelar sistemas de produção industrial. Concretamente, o mestrando Tamatiá Colmán desenvolve um modelo de uma célula de manufatura com vários sensores, atuadores e buffers utilizando CPN Tools. Graças a CPN Tools, permite-se achar e quantizar os pontos de estrangulamento do sistema (gargalos), assim como avaliar o desempenho, achar concorrências, possíveis condições de exclusão mútua e mais eventualidades. Posto em prática, a construção do sistema real permitiria não só comparar os resultados obtidos com a simulação de CPN Tools, como também recolher e analisar dados do sistema em tempo real, fazendo uso da interface de comunicação desenvolvida neste projeto.

1.2. Objetivos

A seguir são apresentados os objetivos propostos pelo CRTi:

"*O objetivo do trabalho é viabilizar de forma simplificada a comunicação entre o software CPN Tools e o ambiente externo. Para isso deverá ser feito o estudo de viabilidade utilizando as linguagens de programação Java, C, C# e também o protocolo TCP/IP, além do estudo aprofundado do conceito de CPN e do software CPN Tools.*

De acordo com o resultado do estudo deverá ser desenvolvido uma (ou mais de uma) interface entre o CPN Tools e um aplicativo em Java, C ou C# (preferência C#) e também uma interface para comunicação TCP/IP.

Para cada tópico desenvolvido, um exemplo prático deve ser apresentado. Em principio esses exemplos devem:
Comunicar uma CPN com um aplicativo em C#.
Comunicar uma CPN com um aplicativo remoto utilizando o protocolo TCP/IP. "

O estudo aprofundado das CPN e do software CPN Tools está descrito na seção 2.1. O estudo de viabilidade do uso das linguagens de programação é descrito a seguir.

1.3. Estudo de viabilidade

1.3.1. Análise da linguagem C

Oferece uma grande possibilidade, já que existe uma biblioteca pronta para gerenciar a conexão como CPN Tools, descrita em (1). Porém a carência de programação orientada a objetos nesta linguagem dificulta a implementação, assim como o tratamento de ambiente gráfico.

1.3.2. Análise da linguagem Java

As possibilidades com a linguagem Java são bem maiores do que com a linguagem C tendo esta, grandes possibilidades com ambientes gráficos, graças aos elementos framework. Além disso, já foram implementadas soluções de comunicação com esta linguagem, e ficam em aberto sendo um claro exemplo a CPN dos *Dining Philosophers* (2).

1.3.3. Análise da linguagem C#

A linguagem C# é uma mistura de Java e C++, além de adicionar elementos novos como recursos de gerenciamento de eventos e a facilidade de manipulação de gráficos entre outras características.

1.3.4. Conclusão

Sendo possível o desenvolvimento da aplicação nestas três linguagens, esta será desenvolvida em C# se baseando no forte potencial que oferecido pela gerencia de eventos, a facilidade para gerar um ambiente gráfico e a preferência por esta linguagem nos objetivos do CRTi.

1.4. Estrutura do documento

Este trabalho apresenta-se dividido em vários capítulos nos quais se tratam todos os aspectos relativos ao projeto. Ainda mais, inclui-se um anexo que complementam e estendem a descrição fornecida na forma de um manual de uso da interface de comunicação. A seguir mostram-se os diferentes capítulos assim como o apêndice:

1. Introdução: Sendo este capítulo, inclui a motivação do projeto, os objetivos a cumprir, uma análise para determinar que linguagem de programação utilizar e a estrutura do documento.
2. Estado da arte: Aborda um referencial teórico sobre as redes de Petri coloridas, CPN Tools e as possibilidades de comunicação neste âmbito.
3. Desenvolvimento da Interface de comunicação: Detalha as fases do desenvolvimento do projeto: elaboração de requisitos, implementação e estabelecimento das comunicações, e capacidades de monitoramento.
4. Teste de software: Mostra as provas efetuadas para avaliar o correto funcionamento da ferramenta.
5. Aplicações: Apresenta-se um exemplo de aplicação para o uso da Interface.
6. Gestão do projeto: Descreve-se brevemente a distribuição das tarefas temporalmente e listam-se as ferramentas utilizadas na elaboração do projeto, mas que não formam parte dele.
7. Conclusões e perspectivas futuras: Apresentam-se as conclusões com relação aos resultados, além das propostas de trabalhos futuros.
8. Referências bibliográficas: Detalham-se as fontes consultadas.

A. Apêndice: Manual de usuário de CPN Communications Interface: Explica detalhadamente como utilizar a ferramenta desenvolvida.

2. Estado da arte

O trabalho foi desenvolvido para comunicar um programa de análise, simulação e modelagem de Redes de Petri coloridas, por isso procede-se a descrevê-las: sua origem e evolução.

2.1. Redes de Petri

As redes de Petri foram desenvolvidas por Carl Adam Petri, aparecendo pela primeira vez na sua tese de doutorado (3) Petri foi um dos poucos no campo da informática que deu nome a alguma das suas invenções, fato que mostra a importância e influência. A principal vantagem de Petri foi a atribuição dinâmica de recursos em modelagens de sistemas distribuídos e concorrentes. Desta maneira, é possível adicionar módulos a um sistema (arquitetura extensível) sem ter que reiniciar de novo a compilação para a distribuição dos recursos (4). Porém isso tinha um preço, cada componente adicionado precisava agir de forma autónoma e o sistema trabalharia assíncronamente de forma global. Petri estava planejando o contrário do que estava sendo desenvolvido já que a maioria dos sistemas eram construídos e desenhados para funcionar de forma síncrona. Além de desenvolver uma técnica de modelagem de sistemas distribuídos assíncronos, estabeleceu uma série de requisitos e objetivos:

- Abandonou a idéia fictícia do estado global e se aproximou mais às leis da física considerando que uma ação discreta normalmente não afeta todos os componentes do sistema, mas só alguns deles. Petri sugeriu que a ocorrência das ações não deveria estar ordenada em forma temporal, mas em relação às causas e efeitos que as comandam, originando a concorrência.
- Baseado na tradição científica, sugeriu que os modelos informáticos deveriam ser reversíveis, ou seja, que a transição de um estado E_1 a E_2 poderia ser também re-calculada para passar de E_2 a E_1.
- A técnica de modelagem não deveria somente ser adequada para descrever o comportamento do sistema, mas também o uso pragmático dos sistemas computacionais pelos humanos. Petri incorporou também numerosas representações gráficas para desenhar os modelos já definindo o que seria os lugares (*Place*s) que mais tarde derivariam nos estados e nas transições.

Esses temas expostos estavam adiantados a esse tempo já que o sincronismo governava os computadores nos anos 60 e ninguém pensava em redes de computadores. Isso provocou que o trabalho de Petri não chamara a atenção imediatamente. Em consideração, os primeiros que utilizaram o seu trabalho foram A. W. Holt em Nova York, escrevendo dois relatorios (5) e (6) que mostravam como as RdP poderíam ser aplicadas na modelagem e análise de sistemas concorrentes. Eles formariam parte do projeto MAC (Multiple Access Computer) do MIT que ajudaria a divulgar a obra e o legado de Petri. Como se relata em (4) não está demonstrado, porém, Petri possivelmente tenha influenciado a representação gráfica em desenho de software UML.

As RdP são utilizadas para modelar o comportamento dinâmico de sistemas discretos. Informalmente, pode-se dizer que elas são formadas por três tipos de elementos: os lugares (*Place*s), as transições (*Transitions*) e os arcos (*Arcs*). Estes ligam os lugares (que representam os estados), às transições (que representam os eventos) que, quando são disparadas provocam a mudança do estado dos sistemas.

Definição informal

A seguir, vai-se fornecer uma definição informal das RdP baseada em (7). As RdP compreendem tanto um ponto de vista estático como dinâmico. Do ponto de vista estático, definem-se os seguintes elementos.

- Lugares (*place*s): Eles representam objetos ou estados. Sua representação gráfica é em forma de elipse.
- *Token*s ou marcas: Representam o valor de um lugar ou a ocorrência do mesmo (lembre-se que os lugares podem representar um estado). Graficamente apresenta-se com uma marca circular (●). Se nas RdP ocorrerem normalmente situações nas quais mais de um *token* se situa num lugar, pode-se representar só com um círculo com o número de *token*s que ocorram nesse momento e nesse lugar.
- Transições (*transitions*): Representam ações nas quais os *token*s mudam de lugar. Representam-se normalmente em forma de retângulo, embora também sejam demonstradas de uma maneira simplificada com uma linha perpendicular ao arco.

- Arcos (arcs): São as ligações entre um lugar e uma transição, ou vice-versa. Definem o caminho a ser percorrido pelos *token*s. Em RdP de maior nível os arcos possuem outra propriedade chamada peso do arco. O peso do arco de entrada de uma transição define o número de *tokens* necessários para poder ativar a transição. Se for um arco de saída de uma transição, este define o número de *tokens* que são transladados ao lugar seguinte. Graficamente os arcos são representados por setas que indicam o sentido de movimentação dos *tokens*. O arco pode ter sentido duplo.

A seguir na Figura 2.1 mostram-se duas partes tiradas de uma RdP maior para mostrar em detalhe os elementos anteriormente definidos. Pode-se observar que na parte esquerda, há dois lugares (p_1 e p_2) cujos arcos de saída levam até a transição t_1. Também existe a possibilidade de que a ocorrência de uma transição produza uma bifurcação do arco e leve este até dois lugares, como mostra a saída da transição t_2 aos lugares p_3 e p_4.

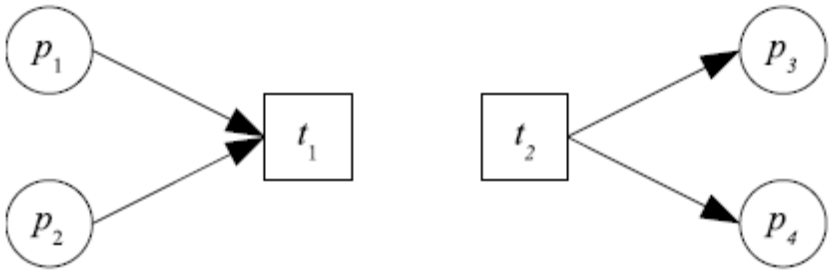


Figura 2.1 Módulos de uma RdP

Desde um ponto de vista dinâmico, há os seguintes fenômenos a serem descritos:

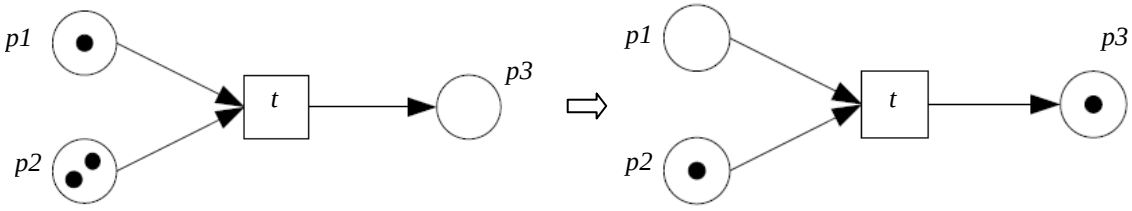


Figura 2.2 Ativação de uma transição

- Ativação ou sensibilização de uma transição: Uma transição é ativada, quando todo o lugar de entrada (conectado a ela) tem igual o maior número de *tokens* que os determinados pelo peso do arco que une o lugar com a transição. Ver a

Figura 2.2 onde os dois lugares anteriores à transição (p1 e p2) contém um e dois *token*s respectivamente. Neste caso, por não estar especificado, o peso de todos os arcos é unitário.

- Execução ou disparo da transição: Uma transição ativada pode executar-se. Quando isto acontece, os *token*s do(s) lugar(es) de entrada da transição são transladados ao(s) lugar(es) na saída da transição. Observa-se como na Figura 2.2 os dois lugares de entrada da transição possuem um e dois *token*s cada um. O arco do segundo só aceita um *token* por passo (peso unitário). Depois que a transição for executada, um *token* é retirado de cada lugar p1 e p2 e um *token* é adicionado no lugar p3. Na fase final, a transição não será ativada já que só há *token* em um dos lugares de entrada.
- Uma transição ativada pode não ser executada (já que a própria transição pode possuir uma condição de execução), mas uma transição não ativada não pode ser executada.
- O comportamento dinâmico de uma RdP não e determinístico. A seqüência de passo das transições também não é.
- Diz-se que duas transições estão em conflito se seu lugar de entrada é comum. Nesse caso, deverá ser a instrução do arco quem determina qual transição deve ser ativada. Um *token* de um lugar que participa no conflito não pode ser usado por duas transições ao mesmo tempo.
- A execução de uma transição com diferente número de lugares de entrada e de saída implica a mudança do número global dos *token*s da RdP

Definição formal

Em relação à definição formal podem-se consultar as seguintes obras para maiores detalhes além da definição aqui exposta: (7), (8), e (9).

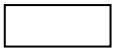
Uma RdP é uma quádrupla RdP = (E, T, F, M_0) que define um grafo bipartido, dirigido e conexo, onde:

1. E é o conjunto finito de estados $E = \{e_1, e_2,..., e_n\}$
2. T é o conjunto finito de transições $T = \{t_1, t_2,..., t_m\}$
3. F é o conjunto finito que contem as relações de fluxo que definem como se dá a transformação de um estado em outro pela ocorrência das ações no sistema.

Portanto, possui as relações que unem um estado a uma transição e as que unem uma transição a um estado. Esta relações de fluxo vem representadas pelo conjunto $F = \{ (x,y) \in E \times A \cup A \times E \}$

4. M_0 é o conjunto finito que define a marcação inicial dos estados $M_0 = \{m_{01}, m_{02}, ..., m_{0n}\}$

As RdP possuem as seguintes características:

a. $E \cup A \neq \emptyset$ - o grafo não é vazio e não tem elementos isolados.
b. $E \cap A \neq \emptyset$ - seus conjuntos de estados e de ações são disjuntos.
c. Existe uma notação gráfica das RdP que determina que:
    - Os estados são representados por elipses.
    - As ações são representadas por retângulos. 
    - Os elementos de representação de fluxo são setas ou arcos.

Ao longo da evolução das RdP apareceram diferentes tipos, mas o passo mais importante aconteceu quando apareceram as RdP de alto nível nos anos 70. Estas se diferenciam das outras no *token*, que até esse momento podia ter um estado definido por um número inteiro ou na maioria dos casos até por um booleano. Com as RdP de alto nível, os *token*s são objetos complexos definidos arbitrariamente, que podem descrever tanto o estado de um processo complexo como uma base de dados (9). Kurt Jensen compara esta evolução com o surgimento das linguagens de alto nível a partir das linguagens montadoras. Embora os dois tipos de linguagens (montadoras e orientados a objetos) tenham o mesmo poder expressivo, as linguagens e as RdP de alto nível permitem uma estruturação melhor e métodos abstratos mais adequáveis à complexidade de grandes sistemas (10). Um destes tipos de RdP de alto nível é a Rede de Petri Colorida.

Exemplo de RdP:

Um elevador onde os andares são representados pelos andares e as transições são ativadas pelos botões do elevador para subir o baixar.

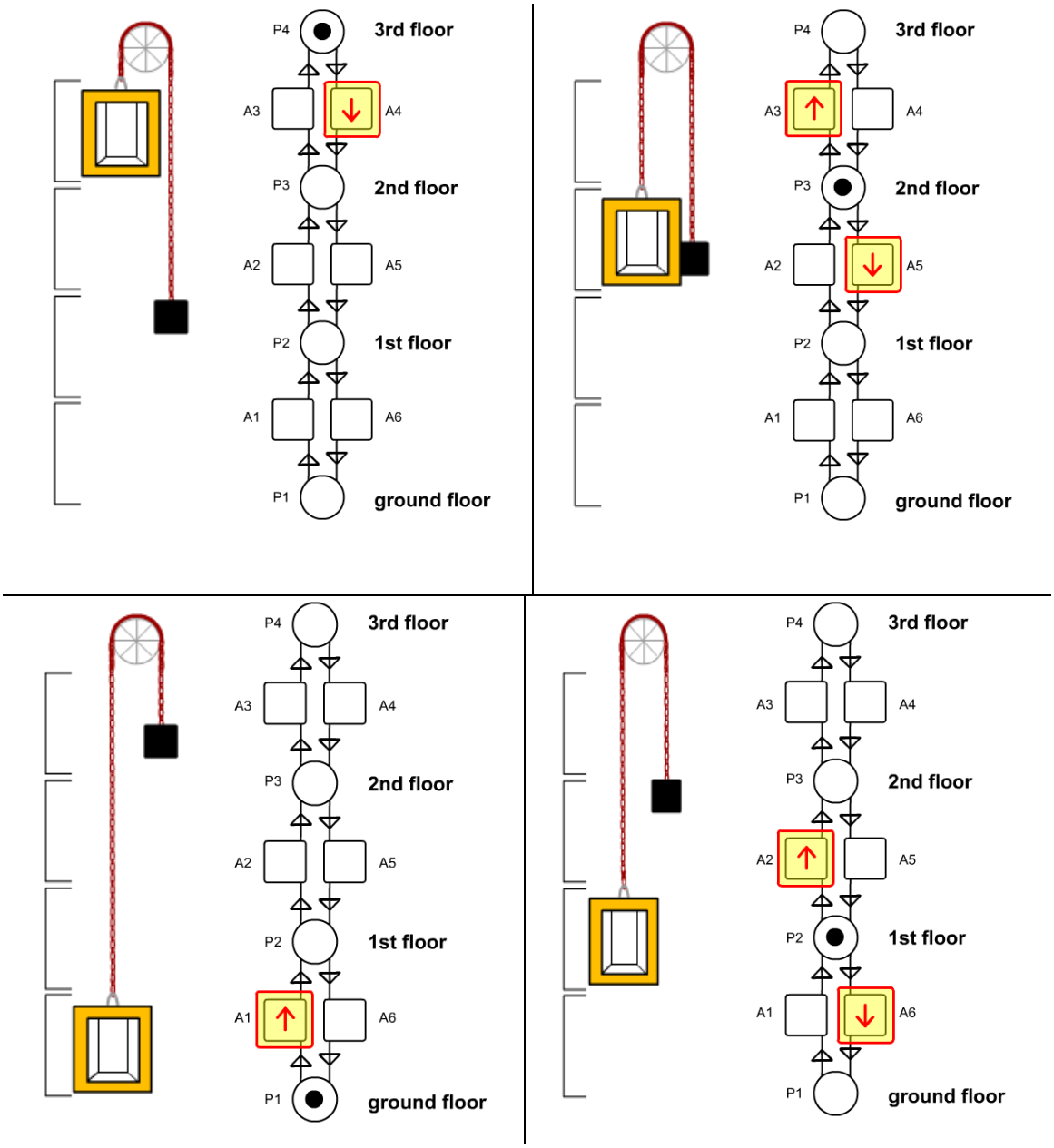


Figura 2.3 Exemplo de uma RdP de um elevador

2.2. Redes de Petri coloridas

Sua primeira versão fez parte da tese de doutorado de Kurt Jensen em 1979/80 (11) baseado no trabalho pioneiro de Genrich ou Lautenbach sobre RdP do tipo Predicate/Transition (12). O termo Colored Petri Nets (CPN) apareceu pouco depois em um artigo publicado também por Jensen (13). A partir daí, o departamento de computação da Universidade de Aahrus, comandado atualmente por Jensen, tem continuado a desenvolver e a consolidar o modelo básico das CPN, adicionando extensões à definição (tendo em conta o tempo e a hierarquia) e desenvolvendo aplicações de modelagem, simulação e análise das CPN.

O principal valor adicionado pelas CPN é a individualização dos *token*s, atribuindo cores diferentes a eles, podendo assim representar diferentes processos, recursos ou dados complexos dentro de uma mesma rede. As cores representam distintos tipos de dados complexos ou simples, (por exemplo, enumeração ou booleano) segundo a sua prévia declaração. Cada lugar pode conter um ou mais *token*s, sendo cada um deles de um tipo de dado associado (cor). Finalmente, este lugar leva especificada a coleção de cores (*colour set*) que pode nele ser contido. Assim, os lugares com seus *token*s representam o estado do sistema. Além do mais, os arcos possuem sua expressão a qual permite transladar a coleção de cores de maneira parcial, (se o *colour set* for composto) inteira ou transformando-a. Esta também pode ser uma expressão lógica a ser processada.

Segundo (14), as CPN são compostas de três diferentes partes: estrutura, declarações e inscrições.

Estrutura é um grafo dirigido com dois tipos de vértices (lugares e transições). Os lugares são representados por elipses e as transições por retângulos. Declarações compreendem a especificação dos conjuntos de cores e declarações de variáveis. As inscrições são diferentes dependendo do componente da rede. Os lugares possuem três tipos: nome, conjunto de cores e expressão de inicialização (initial marking). As transições possuem o nome e a expressão guarda (guard) e os arcos têm sua expressão.

A continuação procede-se a dar uma definição formal das CPN. A definição formal é precisa já que ela evita a ambigüidade e fornece uma descrição mais completa.

Para isso, definir-se-ão (baseado em (15) e (1)) os conceitos de multiset, CPN não hierárquica e CPN hierárquica.

2.2.1. Definição formal

Multiset

Um *multiset* é um conjunto que agrupa vários elementos, e atribui um número em função ao número de vezes que aparece o mesmo elemento. Por exemplo, temos o multiset m_B seguinte:

$$m_B = 1`(1, "COL") + +2`(3, "ORI") + +3`(2, "DAS")$$

Este multiset contem 1 elemento de tipo product formado por o inteiro 1 e a *string* "COL", dois formados por o inteiro 3 e a *string* "ORI" e três elementos formados por o inteiro 2 e a *string* "DAS". Outra representação equivalente seria a seguinte:

$$m_B(s) = \begin{cases} 1\ se\ s = (1, \text{"COL"}) \\ 2\ se\ s = (3, \text{"ORI"}) \\ 3\ se\ s = (2, \text{"DAS"}) \\ 0\ nos\ outros\ casos \end{cases}$$

Formalmente, os *multiset* se definem como segue:

Um *multiset m* sobre um conjunto não vazio *S* é uma função:

$$m \in [S \to \mathbb{N}].$$

O número não negativo *m(s)* é também chamado de coeficiente de *s* em *m.* Em geral o *multiset* vem definido formalmente como:

$$m = \sum_{s \in S} m(s)`s$$

O elemento $s \in S$ é um membro de um *multiset m* se $m(s) \neq 0$. Nesse caso diz-se também que $s \in m$.

Adição, multiplicação escalar, comparação e tamanho definem-se como segue, considerando que m_1 e m_2 e m são *multisets*, e $n \in \mathbb{N}$.

1. $\forall s \in S: (m_1 + +m_2)(s) = m_1(s) + m_2(s)$

2. $\forall s \in S: (n ** m)(s) = n * m(s)$
3. $m_1 <<= m_2 \Leftrightarrow \forall s \in S: m_1(s) \leq m_2(s)$
4. $|m| = \sum_{s \in S} m(s)$

Para definir as CPN é preciso definir as seguintes funções:

- *Type(v)* é o tipo de dado de uma variável *v* e analogamente *Type(expr)* é o tipo de dado resultante da expressão *expr*.
- O conjunto de *multisets* de um conjunto *T* será representado como T_{MS}.
- O tipo de dado Booleano vem representado com $\mathbb{B} = \{true, false\}$

2.2.2. CPN não hierárquica

Uma CPN não hierárquica é um nônuplo CPN = (*P,T,A,Σ ,V,C,G,E, I*) onde:

1. *P* é um conjunto finito de lugares (*Place*s)
2. *T* é um conjunto finito de transições (*Transitions*)
3. $A \subseteq P \times T \cup T \times P$ é um conjunto de arcos direcionados.
4. *Σ* é um conjunto finito de *colour set*s não vazios.
5. *V* é um conjunto finito de tipos de variáveis tais que $Type[v] \in \Sigma$ para todas as variáveis $v \in V$.
6. $C : P \longrightarrow \Sigma$ é uma função de *colour set* que atribui um *colour set* a cada lugar.
7. $G : T \longrightarrow EXPR_V$ é uma função guarda que atribui uma expressão guarda (*guard*) a cada transição tal que $Type[G(t)] = Bool$.
8. $E : A \longrightarrow EXPR_V$ é uma função de expressão de arco que atribui uma expressão de arco a cada arco *a* tal que $Type[E(a)] = C(p)_{MS}$ onde *p* é o lugar conectado ao arco *a*.
9. $I : P \longrightarrow EXPR_V$ é uma função de inicialização que atribui uma expressão de inicialização (*initial marking*) a cada lugar *p* tal que $Type[I(p)] = C(p)_{MS}$.

As CPN permitem ser analisadas de forma interativa ou automática. A primeira é útil para poder depurar passo a passo, podendo mudar o modelo e comprovar que o modelo comporta-se da forma esperada, e daí o nome de interativa. A forma automática serve para testar e para gerar análises de desempenho. Para testar introduzem-se os

critérios de parada apropriados, e para as análises de desempenho, se introduzem coletores de dados que geram os correspondentes informes. Com isso, os seguintes conceitos que possibilitam o disparo de uma transição são definidos e assim, a dinâmica do sistema. Previamente define-se por *EXPR*(*e*)<*b*> a avaliação da expressão (*EXPR*) no elemento da CPN *e* relativa à ligação *b*.

Para uma CPN = (*P*,*T*,*A*,*Σ*,*V*,*C*,*G*,*E*,*I*) definem-se os seguintes conceitos:

1. Uma marcação (*marking*) é uma função *M* que atribui a cada lugar $p \in P$ um *multiset* de *token*s $M(p) \in C(p)_{MS}$.
2. A marcação inicial (*initial marking*) M_0 é definida por $M0(p) = I(p)\langle\rangle$ para todo lugar $p \in P$ (<> denota uma ligação vazia).
3. As variáveis de uma transição *t* vêm definidas $Var(t) \subseteq V$ e são as variáveis livres que aparecem na guarda de *t* e nas expressões dos arcos ligadas a ela.
4. Uma ligação (*binding*) de uma transição *t* é uma função *b* que atribui a cada variável $v \in Var(t)$ um valor $b(v) \in Type[v]$. O conjunto de todas as ligações de uma transição *t* denomina-se *B*(*t*).
5. Um elemento de ligação (*binding element*) é um par (*t*, *b*) tal que $t \in T$ e $b \in B(t)$. O conjunto de todos os elementos de ligação *BE*(*t*) de uma transição *t* é definido $BE(t) = \{(t,b) | b \in B(t)\}$. O conjunto de todos os elementos de ligação de uma CPN denomina-se *BE*
6. Um passo (*step*) $Y \in BE_{MS}$ é um *multiset* não vazio e finito de elementos de ligação.

Para que aconteça o passo de um estado (lugar) a outro, precisam-se das seguintes condições: Avaliação positiva da guarda e a existência dos *token*s requeridos pelas expressões de arco que vão desde o lugar à transição correspondente. Nota-se que as avaliações produzem-se uma por vez e de maneira aleatória.

Um passo (*step*) $Y \in BE_{MS}$ é habilitado em uma marcação M se e somente se:

1. $\forall (t,b) \in Y: G(t)\langle b\rangle (= true)$
2. $\forall\, p \in P: \ {}^{++}_{MS}\sum_{(t,b)\in Y} E(p,t)\langle b\rangle \ll= M(p)$

Se um passo (*step*) $Y \in BE_{MS}$ está habilitado, pode dar-se a ocorrência, dando lugar a uma nova marcação *M*' definida por:

$$M'(p) = \left(M(p) --^{++} \sum_{MS^{(t,b)\in Y}} E(p,t)\langle b\rangle \right) ++^{++} \sum_{MS^{(t,b)\in Y}} E(t,p)\langle b\rangle : \forall\, p \in P$$

Resumindo esta equação, a nova marcação será igual à primeira menos os *token*s consumidos pelos arcos de entrada mais os produzidos pelos arcos de saída.

Exemplo:

A seguinte CPN mostra uma república que há 4 moradores (representados por *per*) e dois pratos para comer. Possui dois *colour sets*: os pratos e os moradores. Como condição os moradores têm que lavar o prato para poder comer. Um desses moradores (*per(3)*) quebra, às vezes, o prato por enquanto que está lavando-o. Conseguintemente chegará um momento, que não haverá mais pratos disponíveis. Inicialmente a casa começa com dois pratos sujos na pilha e os moradores fora da casa. As pessoas podem se encontrar em 5 *places*: Fora da casa "Aluga casa", sem fome, esperando pratos (com fome), lavando prato ou comendo. Os pratos, podem se encontrar ou na pilha (pratos sujos) ou sendo utilizado por uma pessoa quando estão sendo lavados ou se a pessoa está comendo.

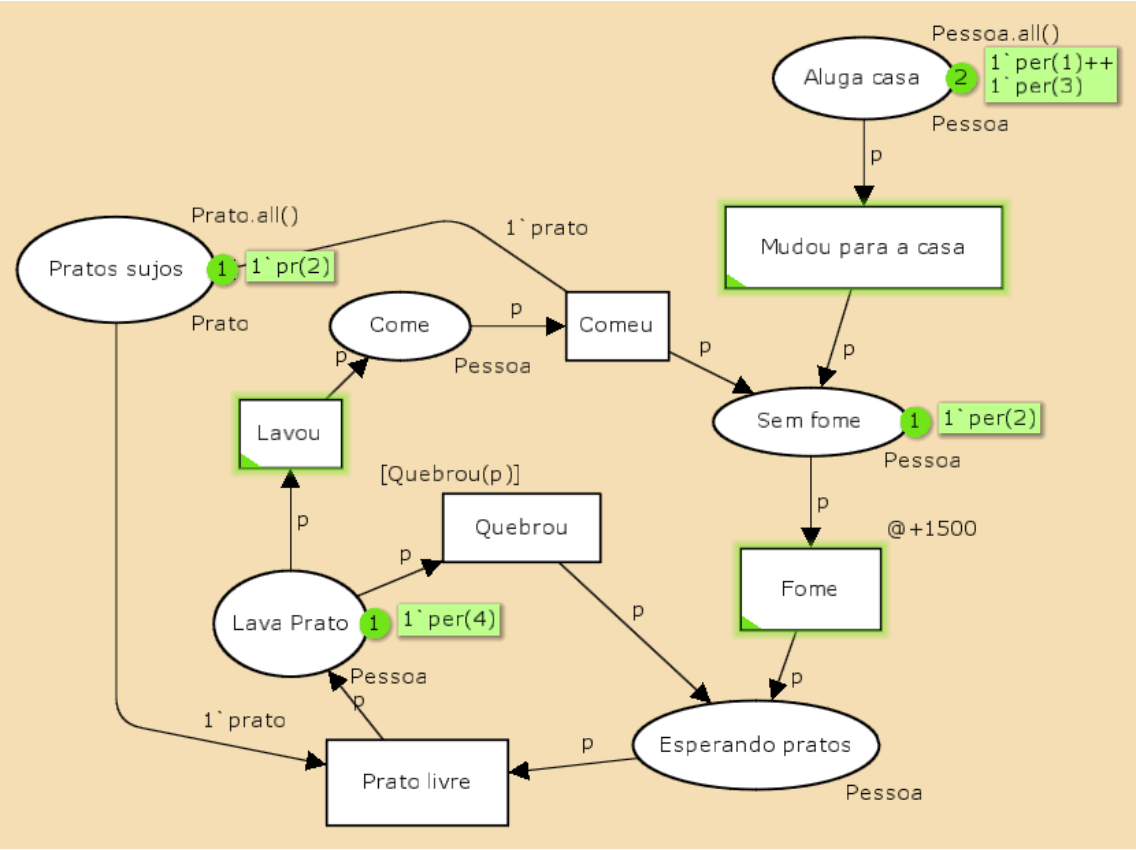


Figura 2.4 CPN exemplo de república

2.2.3. CPN hierárquica

As CPN modelam sistemas de processos grandes e complexos. Assim se fossem representados todos os estados de um sistema grande em uma CPN, acabar-se-ia perdendo visibilidade e a perspectiva do modelo. Além disso, o modelador precisa concentrar-se em poucos detalhes ao mesmo tempo e, conseqüentemente, é melhor o uso de módulos. Por último, se um módulo vai ser utilizado várias vezes é útil usar este tipo de configuração, permitindo que a edição ou as modificações feitas afetem em todo lugar onde for usada. Para evitar isso existe a hierarquia nas CPN, podendo-se construir módulos ou sub-CPNs (*pages*) as quais podem conter CPN normais ou ao mesmo tempo sub-CPNs hierárquicas. Existem cinco construtores de hierarquia (15):

- Substituição com porta
- Substituição com socket
- Substituição de tipo assignment

- Invocação
- Fusão

Sem entrar em detalhes sobre a definição das RdP hierárquicas (Ver (15) para mais informações), encontra-se abaixo o exemplo de uma RdP (Figura 2.5) na qual aparece uma subrede, cujo conteúdo aparece definido na Figura 2.6. Na segunda figura podem ser observadas as transições e estados entre o estado *p1* e *p2*.

Figura 2.5 CPN hierárquica

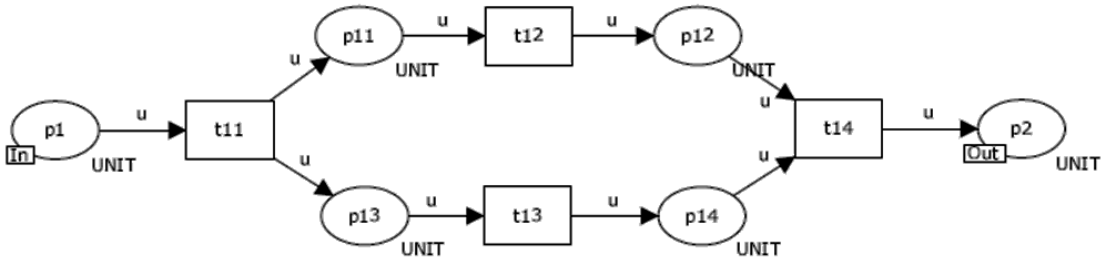


Figura 2.6 Detalhe do subproceso

2.3. CPN Tools

CPN Tools é uma ferramenta desenvolvida pelo departamento de ciências de computação da Universidade de Aarhus na Dinamarca que consiste em uma interface que permite a edição, simulação e análise das CPN utilizando um ambiente gráfico do tipo *post-WIMP* (17), e surge como redesenho da anterior aplicação desenvolvida também por eles Design/CPN. CPN Tools. Mantido pelo CPN Group da Universidade de Aarhus, está protegido por uma licença definida por eles (18), mas é gratuita. Atualmente mais de 9000 organizações em todo o mundo, delas aproximadamente um 6% comerciais, possuem uma licença. No caso do Brasil 487 organizações (14 comerciais) utilizam CPN Tools.

2.3.1. Biblioteca Comms/CPN

A biblioteca foi desenvolvida por Guy Gallasch e Lars M. Kristensen no departamento de Engenharia de Sistemas de Computação da University of South Australia e permite o envio de informações entre processos e aplicações externas mediante o protocolo TCP/IP. Esta biblioteca foi apresentada no congresso sobre CPN da Universidade de Aarhus de 2001 (19). Esta biblioteca cumpre os seguintes requisitos:

- Iniciar conexões de saída com aplicações externas.
- Aceitar conexões de entrada com aplicações externas.
- Enviar e receber dados de qualquer tipo (polimórficos) a uma aplicação externa.
- Fechar conexões com aplicações externas.
- Ser capaz de tratar com conexões múltiplas e simultâneas.
- As transmissões devem ser feitas sem erros.

O protocolo TCP/IP é o mais usado, e quase todas as linguagens de programação fornecem bibliotecas para comunicar usando esse protocolo. A arquitetura desta biblioteca (Figura 2.7) baseia-se em três camadas: uma que utiliza o método de comunicação básico e implementa um interface TCP/IP, outra que trabalha com a transmissão de dados, e outra que gerencia as diferentes conexões que houver. O propósito explicado em (19) relata que são fornecidas duas opções de comunicação: uma TCP pura que conecta diretamente a aplicação predecessora de CPN Tools (chamada Design/CPN) relacionada com a camada de comunicação (Communication Layer), e outra opção de comunicação tipo Messenger. Esta é citada no site de suporte da biblioteca (2) como a melhor opção a utilizar já que oferece identificador da conexão, controle sobre o sucesso da transferência e segmentação das transferências em pacotes de dados de tamanho e codificação customizáveis.

2.3.2. Camada de comunicações

Esta camada fornece a interface para a comunicação básica com uso do protocolo TCP/IP. Cumpre os requisitos previamente mencionados além de ser capaz de gerenciar as múltiplas conexões e também não identifica o tipo de dado que estiver sendo transferido (unicamente transmitem-se *byte*s). Como se pode ver na Figura 2.8 não se inclui nenhum identificador para uma possível conexão múltipla, nem referencia ao tipo de dado a transferir.

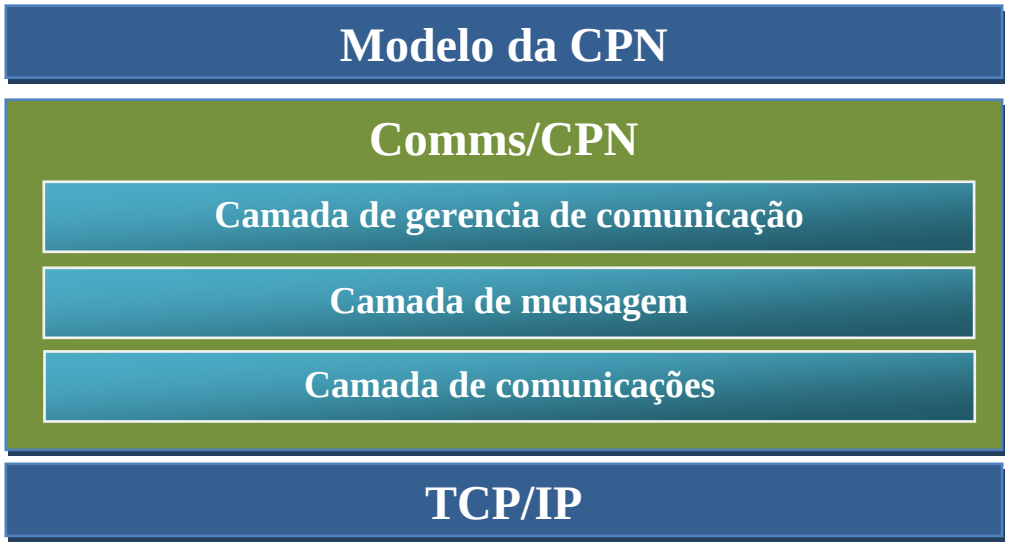


Figura 2.7 Arquitetura da biblioteca Comms/CPN

```
signature COMMS_LAYER =
	sig
		type channel
		exception BadAddr of string

		val connect : string * int -> channel
		val accept : int -> channel
		val send : channel * Word8Vector.vector -> unit
		val receive : channel * int -> Word8Vector.vector
		val disconnect : channel -> unit
	end;
```

Figura 2.8 Signature da camada de comunicação em SML

2.3.3. Camada de mensagem

O objetivo da camada de mensagem é fornecer uma abstração para a troca de mensagens de forma fidedigna, já que a fornecida pelo TCP/IP não é suficiente. Quando se tem diferentes tipos de dados, é necessário segmentar o envio destes para poder diferenciá-los e controlar o correto envio dos mesmos. O signature que descreve a camada da Mensagem está definido na Figura 2.9 Signature da camada de mensagem em SML.

```
signature MESSAGING_LAYER =
	sig
		type channel
		exception InvalidDataExn of string

		val send : channel * Word8Vector.vector -> unit
		val receive : channel -> Word8Vector.vector
	end
```

Figura 2.9 Signature da camada de mensagem em SML

As mensagens são segmentações de no máximo 128 *byte*s, e o mais significativo se utiliza como *byte* de controle. O bit mais significativo estabelece se a transmissão compreende mais de uma mensagem o não. O resto dos bits deste *byte* mais significativo mostra o tamanho em *byte*s da mensagem (máximo 127 *byte*s de dados).
A Figura 2.10 mostra a distribuição das mensagens.

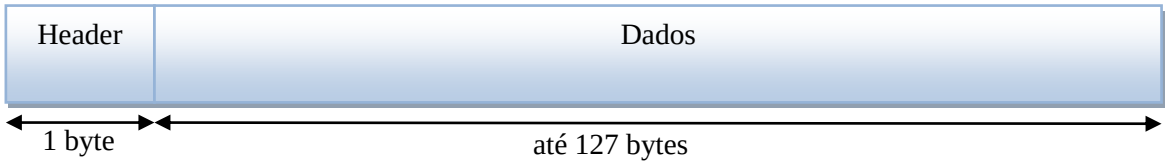


Figura 2.10 Distribuição da mensagem

2.3.4. Camada de gerência de comunicação

Esta camada, junto com a de mensagem, possibilita a execução dos requisitos previamente expostos (na Biblioteca Comms/CPN). Esta camada possui as seguintes características: Identificação da conexão mediante uma string característica, transmissão de dados polimórficos e codificação e decodificação adaptável aos distintos tipos de dados considerados. Estas funções se comportam de maneira síncrona, ou seja, bloqueiam o programa até que seja concluído seu desempenho (até receber uma mensagem do exterior ou até receber uma conexão de entrada por exemplo) ou até que uma exceção é ativada. Na Figura 2.11, descrevem-se as funções da camada chamada *Connection Manager*, no entanto, com o posterior desenvolvimento e aperfeiçoamento foi adicionada alguma como função que verifica se uma conexão está aberta e disponível (*canreceive).* Ver mais detalhes em (2).

```
signature CONN_MANAGEMENT_LAYER
	sig
		type Connection
		exception ElementMissingExn of string
		exception DupConnNameExn of string

		val openConnection : Connection * string * int -> unit
		val acceptConnection : Connection * int -> unit

		val send : Connection * 'a * ('a -> Word8Vector.vector) -> unit
		val receive : Connection * (Word8Vector.vector -> 'a) -> 'a

		val closeConnection : Connection -> unit
	end
```

Figura 2.11 Signature da Camada de gerência de comunicação em SML

Mostram-se a seguir as seguintes funções. Para maiores detalhes pode-se consultar a referência (18):

- ConnManagementLayer.openConnection: `string * string * int -> unit`

Estabelece-se uma conexão como cliente tomando como primeiro parâmetro o nome da conexão recém criada, como segundo o nome do servidor ou o endereço IP dele e o número de porta.

- ConnManagementLayer.acceptConnection: `string * int -> unit`

Cria uma conexão como servidor, especificando o nome da conexão recém criada e a porta onde serão aceitas as novas conexões. Esta função bloqueia CPN Tools até que uma conexão de entrada tenha lugar, e assim o servidor termina a possível recepção de conexões de entrada. Se for requerido estabelecer uma nova conexão, é preciso executar de novo esta instrução, tendo o cuidado de não repetir o identificador da conexão, cujo caso ativará a exceção `DupConnNameExn`.

- ConnManagementLayer.send: `string * 'a * ('a -> Word8Vector.vector) -> unit`

Esta é a função de envio. Como primeiro parâmetro, deve ser introduzido o nome da conexão a utilizar, como segundo parâmetro, a variável a enviar e como terceiro, a função de codificação. Posteriormente nesta mesma seção, serão descritas estas funções.

- ConnManagementLayer.receive: `string * (Word8Vector.vector -> 'a) -> 'a`

A correspondente função de recepção deve ser ativada introduzindo o identificador da conexão dado pelo primeiro parâmetro. Além disso, o outro parâmetro será a função de decodificação, que determina o tipo de dado que se terá na saída na própria definição dela.

- ConnManagementLayer.canReceive: `string -> bool`

Esta função comprova se a conexão identificada pela *string* está operativa e devolve um booleano com *true* no caso afirmativo ou *false* no contrário.

- ConnManagementLayer.closeConnection: `string -> unit`

Esta função fecha a conexão identificada pelo primeiro parâmetro.

- *string*Encode *s* e integerEncode *i*:

Estas funções pré-definidas são usadas para a codificação de *string* ou inteiro a *byte* e são utilizadas na função de envio.

- *string*Decode *v* e integerDecode *v*:

Analogamente estas funções decodificam o vetor *v* em *string* ou inteiro e se utilizam na função de recepção.

Estas duas últimas funções de codificação e decodificação são as fornecidas pelo grupo que desenvolveu a biblioteca COMMS/CPN, mas podem ser desenvolvidas funções personalizadas que lidem com qualquer tipo de dado.

Entretanto existem duas exceções que podem ser ativadas nesta biblioteca. Elas estão relacionadas com o uso inapropriado dos identificadores das conexões. Elas são: `DupConnNameExn` quando tenta-se criar uma conexão com um identificador já usado, e a `ElementMissingExn` quando o identificador é desconhecido.

2.4. Comunicação mediante sockets

A comunicação usando Sockets permite a diferença do sistema composto de servidor e receptor TCP e várias conexões usando a mesma porta. A plataforma .NET aloca de maneira otimizada as portas internas diferentes, com a mesma porta comum. Isto também acontece da mesma forma quando se utiliza um navegador, já que é possível utilizar várias abas (ou janelas) embora se esteja usando a mesma porta (porta 80 - HTTP).

Muita documentação considera a comunicação TCP/IP como uma transferência de pacotes que são fragmentados e enviados cada um seguindo seu caminho, e reordenados no destino. Tudo isto é certo, porém o envio de dados não é feito através de pacotes de dados, mas com um *stream*(fluxo) de dados. Existem um *stream* de entrada e outro de saída. Os pacotes de dados que chegam ao *stream* de entrada são reordenados e ficam esperando a leitura do *stream* de entrada. Por isso é importante ter um cabeçalho que indique quantos bytes devem ser lidos do *stream*, por isso na biblioteca de comunicação da CPN Tools (na seção 2.3.3) fala-se de fornecer uma abstração de troca de mensagens. O primeiro byte de cada pacote indica o tamanho do mesmo, para uma correta leitura desde o *stream* de entrada.

Com isso, mesmo que o protocolo TCP/IP assegure que os dados são recebidos na mesma ordem que foram enviados, isto não quer dizer necessariamente que os dados são transmitidos com a mesma fragmentação com a que foram enviados. Por exemplo, se for enviado um dado de 10 *byte*s e seguidamente outro de 8, pode ser que se recebam um *stream* de 10 e outro de 8, mas por causalidade, poderia ter sido também que chegue 14 *byte*s (10 do primeiro e 4 do segundo) e depois os restantes. Por isso será errado processar diretamente os dados quando forem recebidos no *stream* de entrada e conseqüentemente é necessário o conhecimento do tamanho de cada seqüência de dados enviada fazendo o uso do *Message framing.*

Então, não faz sentido comprovar na recepção de uma seqüência de *byte*s que o número de *byte*s recebidos seja igual ao conteúdo do primeiro *byte*. Este tipo de prática errada é comum, normalmente coincide o número de *byte*s recebidos do *stream* com os

enviados originariamente se a rede não tem muita demora nas transações ou quando o volume de dados é pequeno. O procedimento que deve ser acometido usando o primeiro *byte* que indica o tamanho da informação originariamente enviada deve ser unicamente o de ler o número de *byte*s que este indica do *stream* de entrada, mesmo que não coincidam com o número de *byte*s que chegaram ao *stream*.

3. Desenvolvimento da Interface de comunicação

3.1. Requisitos de projeto

REQ1. A linguagem utilizada será C#.

REQ2. Criar uma biblioteca que permita a comunicação complementando as funções de COMMS/CPN (18)

REQ3. Desenvolver um aplicativo que utilize essa biblioteca e que forneça uma comunicação eficaz.

Requisitos de comunicação

REQ4. A comunicação será feita através de Sockets TCP/IP.

REQ5. Será possível realizar múltiplas conexões com o mesmo servidor e com o mesmo cliente e utilizando a mesma porta.

REQ6. Para cumprir com o REQ5, será preciso a adaptação da OSI Layer desde a configuração descrita na Figura 2.10 à configuração da seguinte:

Figura 3.1 Distribuição OSI a implementar

REQ7. Cumprir as condições do *byte* Header para o envio de dados e, na recepção, comprovar que os *byte*s (indicados pelo Header) foram recebidos. O primeiro bit (Not the last packet) serve para detectar se são esperados mais pacotes do dado ou não (no caso do dado a transferir seja maior de 127 *byte*s) e, nesse caso, a transferência seria efetuada de forma fragmentada (Ver seção 2.3.3).

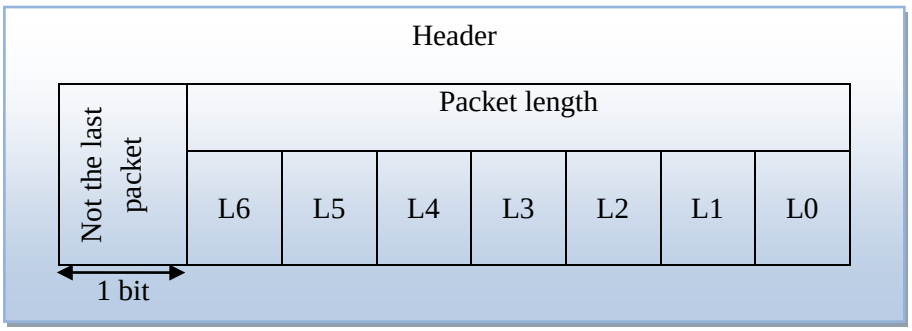


Figura 3.2 Distribuição do byte Header

O *byte* de Controle terá o seguinte conteúdo:

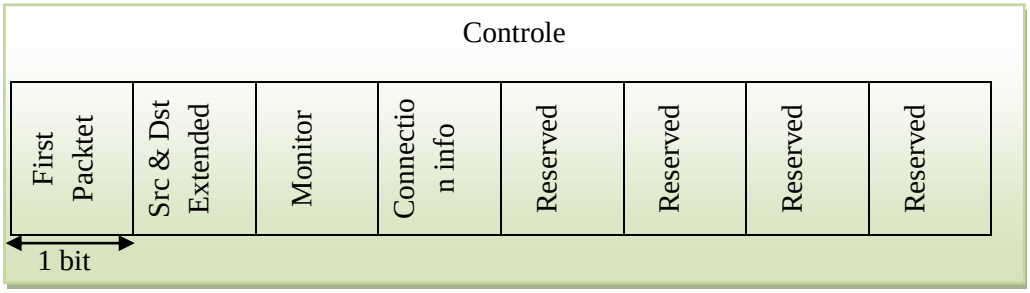


Figura 3.3 Descrição do *byte* de Controle

- *First Packet*: Se este bit estiver ativado, indica que é o primeiro dos pacotes a ser transmitido. Além de isso, indica que o *byte* de Controle e o de *Src&Dst* estão habilitados, ou seja, que o protocolo adaptado está habilitado. Os seguintes pacotes (no caso de uma transmissão fragmentada) não possuirão nem *byte* de Controle nem de *Src&Dst*. Como os dados a transmitir são normalmente codificados em ASCII, nenhum deles supera o número 127 e conseqüentemente não é ativado o primeiro bit do *byte* de Controle.
- *Src & Dst Extended*: A ativação deste bit denota que a origem e o destino da transmissão tenham 1 *byte* cada (até 256 destinações) ou 4 bits cada (até 16 destinações).
- Monitor: A ativação deste bit serve para indicar que o pacote transmitido tem o objetivo de monitorar a atividade da CPN. Estes dados não precisam ser enviados a outro destino, já que meramente tem como propósito monitorar.
- *Connection info*: Quando habilitado, indica que os dados contêm informação da rede recém conectada (número de rede e nome). Essa informação só é transmitida apenas após a conexão ser efetuada.

- O resto dos bits são provisoriamente reservados para um eventual uso futuro. Poderiam ser utilizados para identificar o tipo de dados transferidos (*string*, inteiro ou outros) ou para outros usos.

REQ8. Derivado do requisito REQ5. Para possibilitar o gerenciamento de múltiplas conexões com a mesma porta, o *byte* de Src & Dst (que compreende a informação referente à origem e o destino) irá configurado como segue.

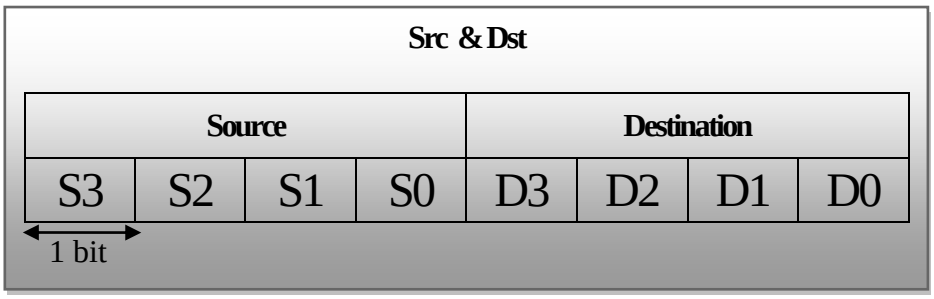


Figura 3.4 Descrição do byte Src & Dst não estendido

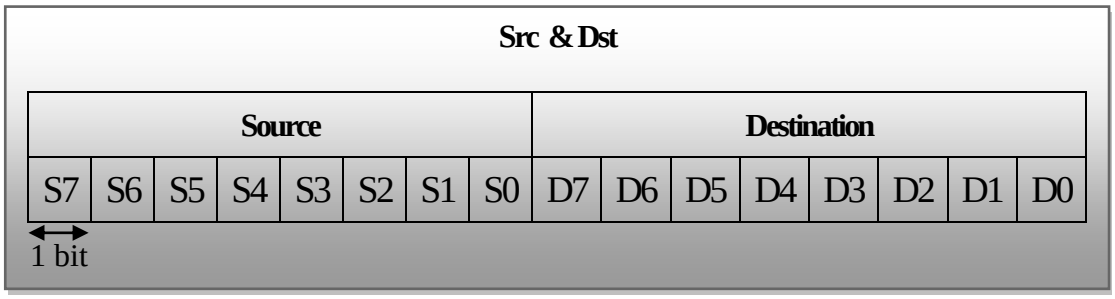


Figura 3.5 Descrição do byte Src & Dst estendido

3.2. Implementação

Nesta seção constam-se exemplos básicos de funcionamento da comunicação com a CPN simples. Posteriormente mostram-se as modificações introduzidas para cumprir com o REQ6 e os outros relacionados com as conexões múltiplas. Mais a frente, será mostrada como foi desenvolvido o monitoramento.

3.2.1. Comunicação simples

A seguir nesta seção descrevem-se comunicações simples de tipo Servidor-Cliente, Cliente-Servidor e Servidor-Cliente-Servidor; sendo o Cliente a aplicação desenvolvida e o Servidor a aplicação CPN Tools.

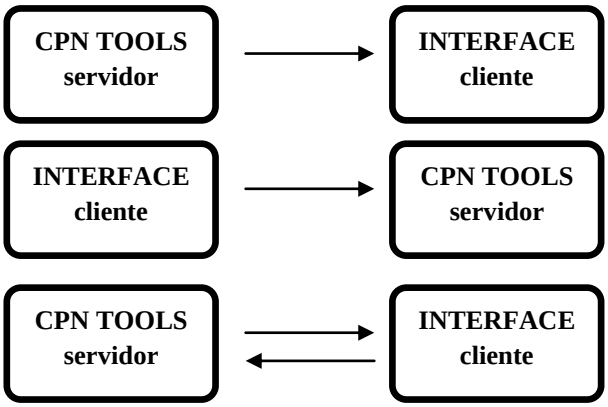


Figura 3.6 Comunicações simples

- Conexão de CPN Tools como servidor ou cliente

O fato de ser servidor ou cliente não afeta a transferência de dados em si, mas somente na hora de efetuar a conexão e a desconexão.

- Recepção de dados provenientes de CPN Tools

Para poder apresentar esta comunicação é criada uma CPN simples, com 3 estados e 3 transições. CPN Tools vai agir como servidor, na criação da comunicação e na iniciativa do envio de dados. Pode-se observar a CPN a seguir:

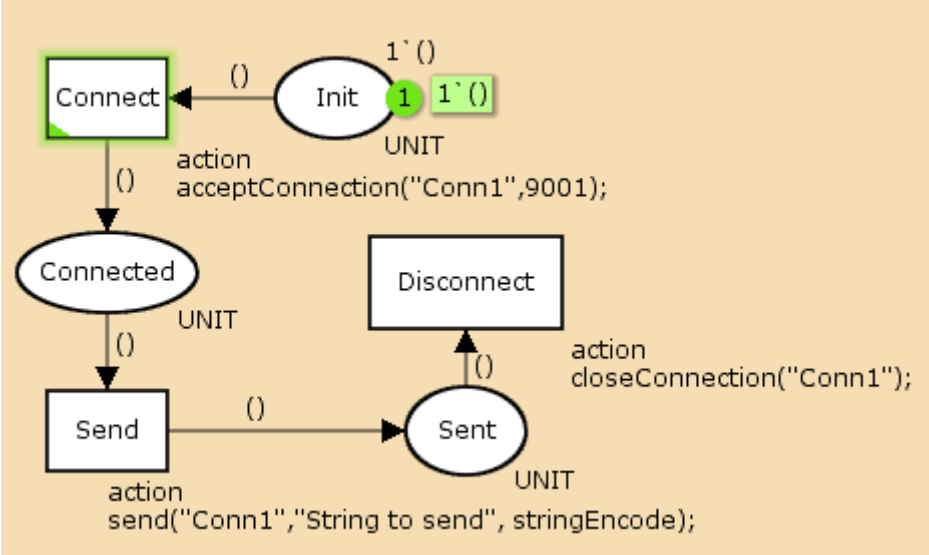


Figura 3.7 CPN para o envio de dados

A CPN apresentada mostra três estados que representam o estado inicial (Init), o estado conectado (Connected) e o estado alcançado quando o envio da *string* “String to send” for feito (Sent). As transições executam as ações de conexão, envio e desconexão.

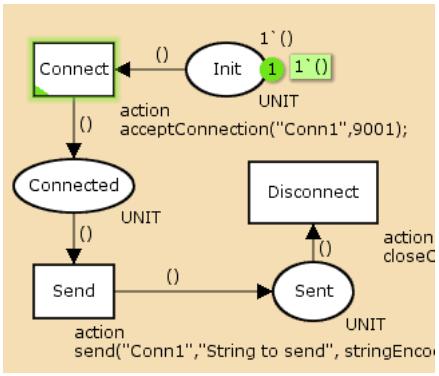


Figura 3.8 CPN na pré-conexão

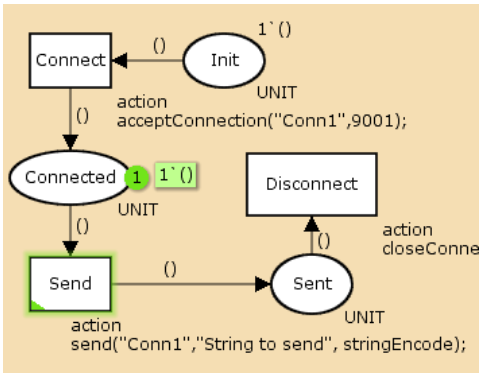


Figura 3.9 CPN no pré-envio

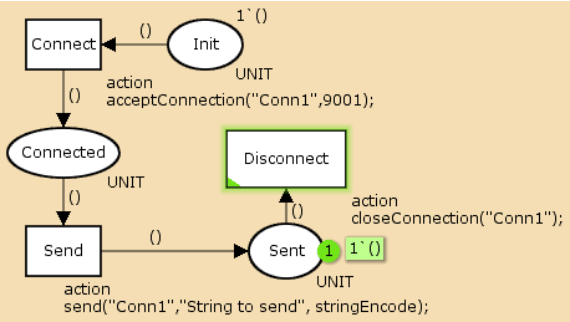


Figura 3.10 CPN na pré-desconexão

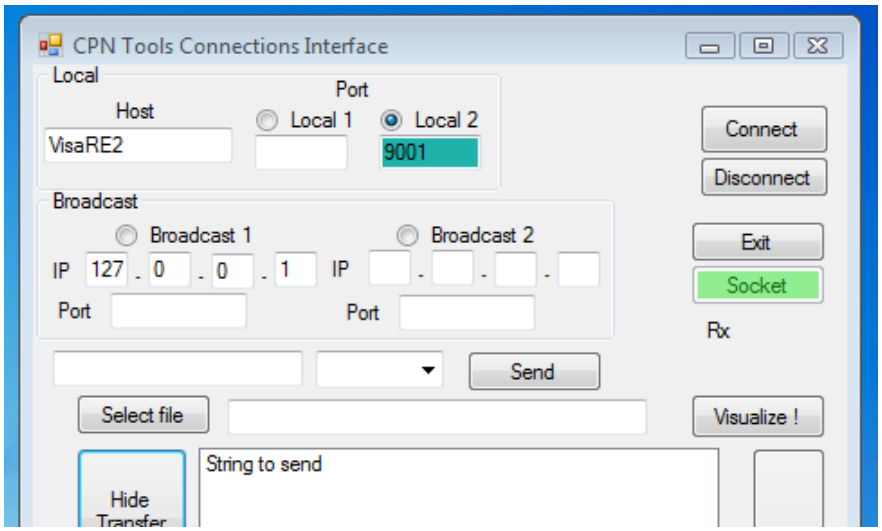


Figura 3.11 Aspecto da aplicação após receber a *string*

Nas Figura 3.8, Figura 3.9, Figura 3.10 e Figura 3.11 mostram-se a seqüência da CPN assim como o aspecto da recepção da *string* enviada.

- Envio de dados para a CPN Tools

No caso de recepção de dados por parte da CPN Tools, procede-se com o uso de uma CPN similar à usada na recepção de dados provenientes da CPN Tools, a qual é mostrada na Figura 3.12.

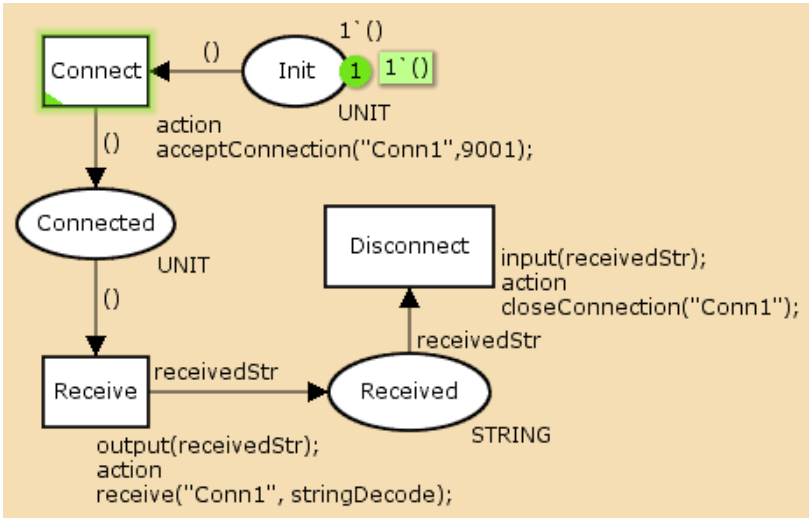


Figura 3.12 CPN para a recepção de dados

Para possibilitar o correto funcionamento da CPN, foi introduzida a variável:

```
var receivedStr : STRING
```

A seguir mostra-se a seqüência de funcionamento desta CPN:

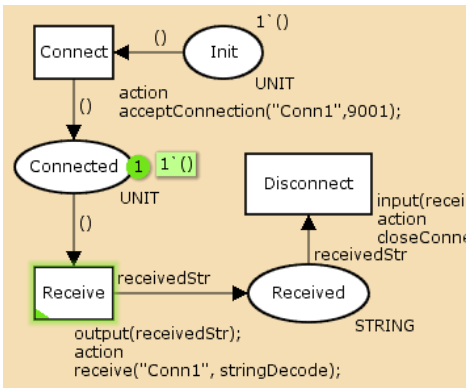


Figura 3.13 CPN na pré-recepção

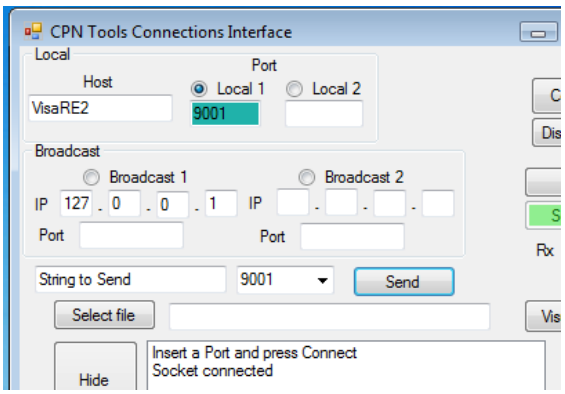


Figura 3.14 Aspecto da aplicação enviando a *string*

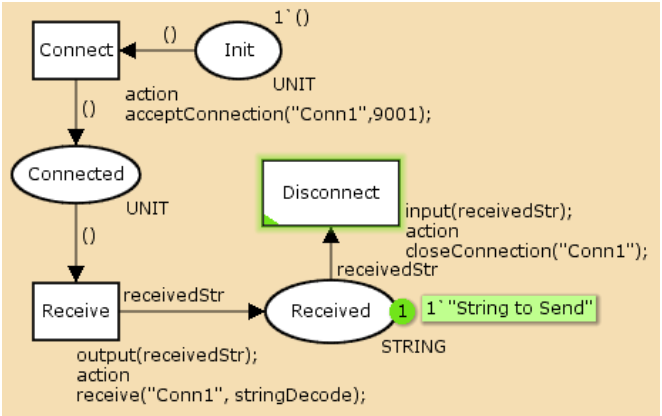


Figura 3.15 CPN após a recepção

- Comunicação bidirecional entre CPN Tools e a interface

Para mostrar a comunicação bidirecional, construiu-se um uma CPN com 4 *places* e 4 transições que se mostra a seguir:

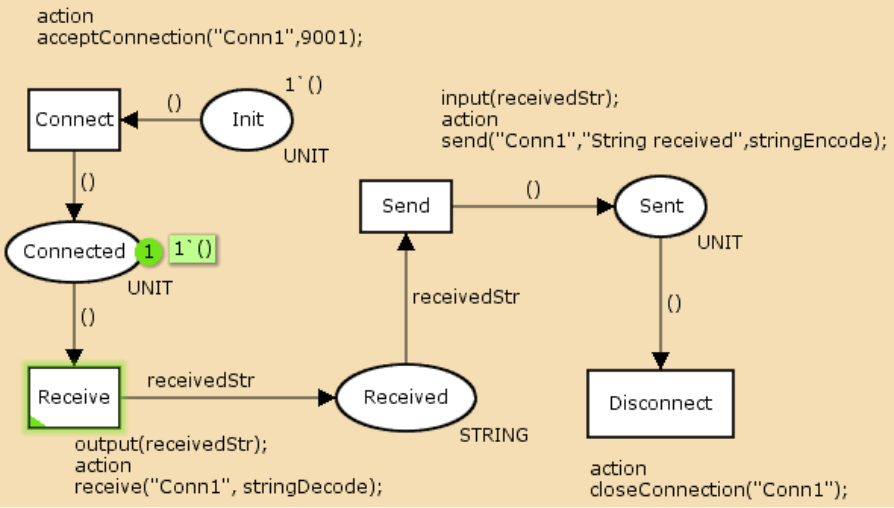


Figura 3.16 CPN com envio e recepção de dados

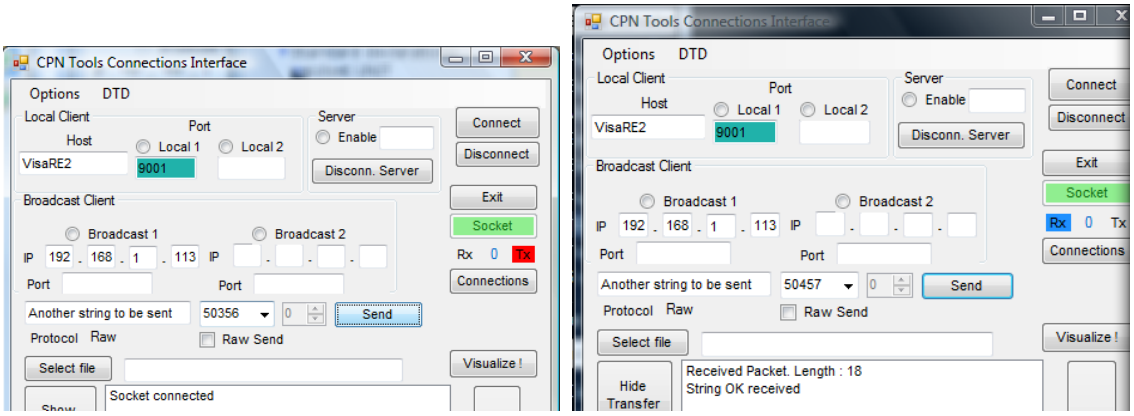


Figura 3.17 Aspecto da Interface durante o envio e a recepção de dados

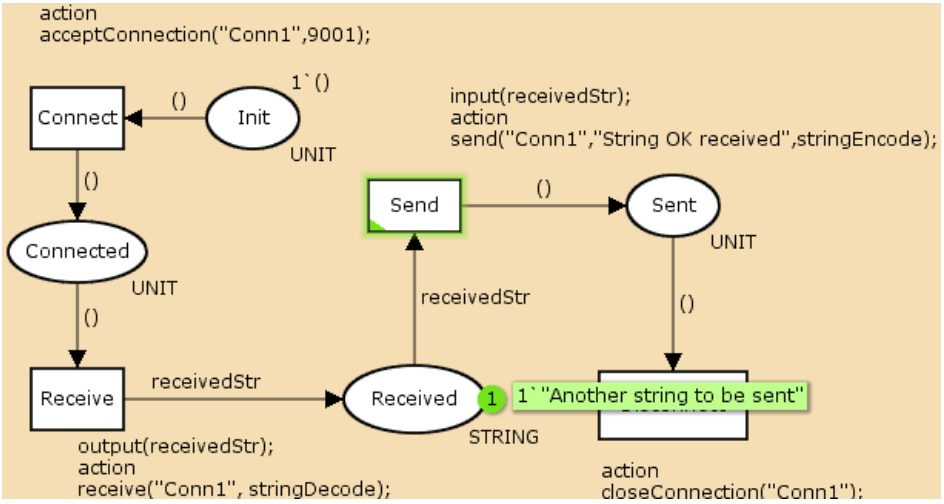


Figura 3.18 CPN após receber uma *string* da interface e antes de enviar outra para a interface.

3.2.2. Comunicação entre várias CPN Tools

Para efetuar a comunicação entre várias CPN ao mesmo tempo é necessário identificar as mesmas. CPN Tools, através da biblioteca COMMS/CPN é capaz de identificar distintas conexões internamente em uma CPN mediante a string da conexão (Ver seção 2.3.4). Porém devido às instruções de comunicação que são feitas de maneira pontual e síncrona, bloqueando todo o sistema até terminar com cada instrução, o que deriva em uma queda de desempenho. Portanto, a melhor opção é construir uma ferramenta que centralize as comunicações entre as distintas CPN (e outros elementos) e que consiga trabalhar de maneira assíncrona e permitindo a multitarefa simultânea. Isto é possível através do protocolo que será implementado na seção 3.2.3 e descrito a seguir.

3.2.3. Comunicação sob Protocolo implementado

As camadas do protocolo já foram descritas nos requisitos REQ6 e nos posteriores. Além dessa informação, o protocolo tem as seguintes características:

- Discriminar o tipo de mensagens transmitidas: de monitoramento, de identificação da conexão e de transferência de dados
- Validar a conexão e evitar possíveis redundâncias nos identificadores.
- Protocolo automático e manual (para seguir a comunicação passo a passo)
- Adapta-se automaticamente para o formato estendido de envio.

Alterações na rede CPN

Para conseguir as funcionalidades anteriormente citadas, e torná-las compatíveis com a aplicação CPN Tools, foram adicionadas as seguintes funções.

- connect: Estabelece e envia a informação da CPN e da conexão.
- connectEncode: Constrói e codifica o pacote a enviar para validar a conexão.
- sendEncode: Constrói o pacote a enviar se adaptando ao protocolo.
- monitorEncode: Similar a sendEncode particularizada para o monitoramento.
- receiveDecode: Decodifica e separa os elementos dos pacotes recebidos.

Além disso, será incluída a variável global que identificará a rede no ambiente exterior.

```
globref NetNumber = 5;
```

É importante distinguir entre este identificador externo, que permite referenciar CPN diferentes (ou outros elementos a serem conectados), e o identificador interno da conexão, que é uma *string* e permite diferenciar várias conexões com o exterior dentro de uma mesma CPN (mesmo identificador numérico). No protocolo implementado, a string identificadora da conexão, será usada unicamente com caráter informativo para o usuário.

3.2.3.1. Função connect

Esta função inicia a conexão, bem como cliente ou servidor, dependendo da opção escolhida pelo usuário. Além disso, envia uma série de dados com informação relativa à conexão. A continuação explica-se brevemente o funcionamento da função de conexão como servidor, que não difere muito da função que conecta como cliente (Ver Figura 3.19 e Figura 3.20)

Primeiramente, abre-se uma conexão como servidor (CPNTools). Quando a conexão cliente (Interface) se conecta a ele, procede-se a enviar a informação do

servidor (CPN Tools) à conexão de entrada. No envio de informação da conexão, vai-se precisar desenvolver uma função que agrupe e codifique esta informação (*string* identificador da conexão interna e identificador numérico da CPN). em um *colset* composto. Esta função (Função connectEncode) será explicada posteriormente .

Esta função recebe como parâmetros de entrada o identificador interno da conexão (ConnName) e a porta da conexão. Primeiramente recopila o número que representa o identificador da rede para a conexão com o exterior da variável global (NetNumber). Depois abre uma conexão como servidor na porta indicada e com o identificador interno da conexão em forma de *string*. Após receber uma conexão satisfatória, acontece a função de envio, efetuada sob a conexão referenciada como "ConnName", que envia o *colset* de tipo *product* composto do nome interno da rede "ConnName" e do número identificador da rede externo. Para possibilitar o envio de um *colset* deste tipo específico é preciso definir a função de codificação correspondente cuja descrição se apresenta a seguir.

```
fun connect(ConnName:string,port:int) =
let
       val Netnum = !NetNumber
       val connection = acceptConnection(ConnName,port)
       val dosend = send(ConnName,(ConnName,Netnum),connectEncode)
in
       (dosend)
end;
```

Figura 3.19 Função connect como servidor

A seguinte função e similar à anterior, recebendo mais um parâmetro de entrada (o IP) e estabelecendo a conexão como cliente.

```
fun connect(ConnName:string,port:int,Ip:string) =
let
       val Netnum = !NetNumber
       val connection = openConnection(ConnName,Ip,port)
       val dosend = send(ConnName,(ConnName,Netnum),connectEncode)
in
       (dosend)
end;
```

Figura 3.20 Função connect como cliente

3.2.3.2. Função connectEncode

Para construir uma função de codificação qualquer, e preciso se referir à definição da função na qual vai ser utilizada. Neste caso, é a função já descrita ConnManagementLayer.send: `string * 'a * ('a -> Word8Vector.vector) -> unit` onde *'a* representa um *colset* polimórfico, e o elemento entre parênteses é a função que transforma esse objeto no vector de *byte*s. No caso da conexão *colset* é de tipo *product*, formado por uma *string* e um número. O *colset* vem definido na seguinte linha, onde é do tipo *product* formado por uma *string* e um inteiro.

```
colset SendData = product STRING*INT;
```

```
fun connectEncode(data2send:SendData) =
let
     val Data = #1 data2send
     val Destination = #2 data2send
     val stringVector= Byte.stringToBytes(Data)
     val extended =if Destination>15 orelse String.size(Data)>15 then 1 else 0
     val Control = Word8.fromInt(128+64*extended+16)
     val SrcDst = if extended<1 then Word8.fromInt(!NetNumber*16 +
       String.size(Data)) else Word8.fromInt(!NetNumber)
     val Dst = if extended>0 then Word8.fromInt(String.size(Data)) else SrcDst
in
     Word8Vector.tabulate(Word8Vector.length(stringVector) +2, fn 0 => Control
     | 1=> SrcDst | n=> Word8Vector.sub (stringVector, n - 2))
end;
```

Figura 3.21 Codificação para a conexão

Primeiramente, como se descreve na Figura 3.21, são extraídos os dois elementos do *colset* por separado. Depois, constrói-se a camada de controle, (de acordo com a Figura 3.3) onde se habilita o bit do primeiro pacote de dados (First Packet) e o da informação da conexão (Connection info). No cabeçalho que indica a origem, codifica-se o número da rede e no de destino, indica-se o tamanho do nome da *String*. Portanto, se o número for maior do que 15 ou se o tamanho do nome em caráteres for maior do que 15 caráteres, automaticamente será habilitado o bit de Src & Dst Extended (no cabeçalho de Controle), alocando então um *byte* (máximo 255) para a origem e outro para indicar o tamanho da *string*. Cabe destacar que a aplicação não deixará alocar um identificador interno para a rede cujo conteúdo precise enviar mais de um pacote, tendo no máximo 124 caráteres.

3.2.3.3. Função sendEncode

Equivalente ao caso anterior, trata-se de uma função de codificação. O parâmetro de entrada é um *colset* do tipo *product* já definido anteriormente e denominado "SendData". Posteriormente, são extraídos os dois elementos: o referente ao destino e colocado na segunda parte do *byte Src & Dst* e na primeira, o número da rede. No *byte* de controle é habilitado o bit *First Packet* e o *Src & Dst Extended* dependendo se algum dos dois números referentes à origem ou ao destino fossem maiores de 15. Ver a descrição na Figura 3.22.

```
fun sendEncode(data2send:SendData) =
let
     val Data = #1 data2send
     val Destination = #2 data2send
     val stringVector= Byte.stringToBytes(Data)
     val extended = if Destination>15 orelse !NetNumber>15 then 1 else 0
     val Control = Word8.fromInt(128+64*extended)
     val SrcDst = if extended<1 then Word8.fromInt(!NetNumber*16+Destination)
      else Word8.fromInt(!NetNumber)
     val Dst = if extended>0 then Word8.fromInt(Destination) else SrcDst
in
     Word8Vector.tabulate(Word8Vector.length(stringVector) +2, fn 0 => Control
     | 1=> SrcDst | n=> Word8Vector.sub (stringVector, n - 2))
end;
```

Figura 3.22 Função sendEncode

3.2.3.4. Função monitorEncode

A função *monitorEncode* é bem parecida à *sendEncode* só que habilita o bit de monitoramento no *byte* de controle e não precisa-se enviar a informação do destino, o que supõe uma poupança de um *byte* a enviar no caso do formato estendido. No caso não estendido, será repetida a origem duas vezes dentro do *byte Src & Dst*.

O monitoramento será inserido na rede automaticamente mediante ao uso de monitores, porém a função mostra-se a seguir na Figura 3.23.

```
fun monitorEncode(data2send:SendData) =
let
     val net = !NetNumber
     val stringVector= Byte.stringToBytes(data2send)
     val extended = if net >15 then 1 else 0
     val Control = Word8.fromInt(128+64*extended+32)
     val SrcDst = if extended<1 then Word8.fromInt(net *16+ net )
      else Word8.fromInt(net )
in
     Word8Vector.tabulate(Word8Vector.length(stringVector) +2, fn 0 => Control
     | 1=> SrcDst | n=> Word8Vector.sub (stringVector, n - 2))
end;
```

Figura 3.23 Função monitorEncode

3.2.3.5. Função receiveDecode

Esta função, que não é estritamente necessária para garantir a comunicação, permite, no entanto, aumentar as possibilidades da mesma. O objetivo desta função é identificar a origem do envio e comprovar que o destino foi correto, mediante ao envio dos cabeçalhos definidos no protocolo (REQ6). Isto permite, por exemplo, seguir caminhos diferentes na CPN receptora segundo a origem dos dados recebidos. Para possibilitar a segmentação dos cabeçalhos que contêm a origem, o destino e para detectar se foram enviadas com formato estendido foram desenvolvidas as funções apresentadas na Figura 3.24 que permitem a manipulação de vectores e listas de números inteiros em SML.

A função *bin(n)* devolve uma lista de uns e zeros mostrando o valor em binário do número introduzido. A função *check(n)* elimina o zero restante adicionado na função anterior no inicio da lista. A função *addZeros(n)* adiciona zeros até completar um *byte* (8 elementos). A função *pow(x,n)* calcula o número x à potência n. A função *bit2Int(l,n)* converte a lista *l* binária criada anteriormente em um número inteiro.

```
fun bin(n) = if n=0 then [0] else [n mod 2]^^ bin(n div 2)
fun check(n) = if List.last(n) = 0 andalso length(n)>1 then
    List.take(n,length(n)-1) else n;
fun addZeros(n) = if length(n)<8 then addZeros(n^^[0]) else n;
fun pow(_,0) = 1 | pow(x,n) = x*pow(x,n-1);
fun bit2Int(l,0) = List.nth(l,0) | bit2Int(l,n) = List.nth(l,n)*pow(2,n) +
    bit2Int(l,n-1);
```

Figura 3.24 Funções para a conversão de dados de tipo *byte*, inteiros e binários

Tendo as funções anteriores prontas, é possível realizar a decodificação do vector de *byte*s a receber discriminando as camadas de controle, de origem e destino. Já

que a função *receiveDecode* vai ser usada como parâmetro na função ConnManagementLayer.receive: `string * (Word8Vector.vector -> 'a) -> 'a`, cabe destacar que o *colset* que vai ter na saída dessa função é chamado *ReceivedTuple* que é de tipo *product* como se mostra a seguir .

```
colset ReceivedTuple = product INT*INT*STRING;
```

A variável *Control* é uma lista binária que representa o *byte* de controle, o que nos permite ter acesso para saber se as camadas de origem e destino são estendidas. Segundo isso obteremos o número da origem e o destino além dos dados recebidos unidos no *colset* de saída como *Src*, *Dst* e *Data*.

```
fun receiveDecode(received) =
let
    val Control=addZeros(check(bin(Word8.toInt(Word8Vector.sub(received,0)))))
    val SecByte = Word8.toInt(Word8Vector.sub(received,1))
    val SrcDst = addZeros(check(bin(SecByte)))
    val extended = List.nth(Control,6)
    val Src = if extended=1 then SecByte else bit2Int(List.drop(SrcDst,4),3)
    val Dst = if extended=1 then Word8.toInt(Word8Vector.sub(received,2)) else
      bit2Int(SrcDst,3)
    val Data = Byte.bytesToString(Word8Vector.tabulate (
        Word8Vector.length ( received) - 2-extended, fn n => Word8Vector.sub
       (received, n+2+extended)))
in
     (Src,Dst,Data)
end
```

Figura 3.25 Função receiveDecode

Esta função tem muito potencial para futuros desenvolvimentos. Poderia ser utilizada para selecionar o arco segundo a origem, para comprovar que o pacote foi recebido na rede certa ou para outros usos.

Na seguinte tabela mostram-se de maneira mais clara as diferentes configurações dos pacotes a enviar. Dependendo se o formato do *byte* de origem e destino é estendido ou não, ocuparão dois *byte*s ou um respectivamente, com a exceção dos pacotes de conexão e monitor.

As funções anteriormente descritas serão incluídas dentro da biblioteca de comunicações que será automaticamente incluída dentro da CPN pela interface.

Nome	Estrutura									Descrição
	Header		Controle							
	Big packet	Packet length	First Packtet	Extended	Monitor	Connexion	Reserved	Reserved	Reserved	Reserved	
Conexão	0	<127	1	X	0	1	X	X	X	X	Informação da CPN
Monitor	0	<127	1	X	1	0	X	X	X	X	Só para monitorar
Curto	0	<127	1	X	0	0	X	X	X	X	Transação de um pacote
Longo 1º	1	127	1	X	0	0	X	X	X	X	Primeiro pacote
Longo meio	1	127	0	X	X	X	X	X	X	X	Pacote intermediário
Último	0	<127	0	X	X	X	X	X	X	X	Último pacote

Tabela 3.1 Descrição das camadas de distintos pacotes

Nome	Controle								Src & Dst								
									Source				Destination				
	First Packtet	Extended	Monitor	Connexion	Reserved	Reserved	Reserved	Reserved	S3	S2	S1	S0	D3	D2	D1	D0	
Conexão não est.	1	0	0	1	X	X	X	X	Número da CPN				Tamanho da *String* da conexão				
Monitor não est.	1	0	1	0	X	X	X	X	Número da CPN				Número da CPN				
Conexão estend.	1	1	0	1	X	X	X	X	Número da CPN				Tamanho da *String* da conexão				
Monitor estend.	1	1	1	0	X	X	X	X	Número da CPN				-				
Nome	Controle								Src & Dst								
									Source								Destination							
	First Packtet	Extended	Monitor	Connexion	Reserved	Reserved	Reserved	Reserved	S7	S6	S5	S4	S3	S2	S1	S0	D7	D6	D5	D4	D3	D2	D1	D0

Tabela 3.2 Descrição especifica dos pacotes de conexão e monitor

3.2.4. Estabelecimento da conexão

Para evitar redundância, no momento de estabelecer a conexão é importante comprovar que a nova conexão não tem um identificador já utilizado. Por isso, a rede CPN envia uma informação própria, que compreende o número da CPN e a *string* identificadora interna da conexão. A Figura 3.26, mostra o diagrama de seqüência da conexão. Neste caso, a CPN atua como servidor e a aplicação como cliente, mas se fosse o contrário não afetaria o mecanismo. O funcionamento é o seguinte: Uma vez que o servidor abre a conexão e um cliente conecta com esse servidor através do *Socket*, a aplicação adiciona uma conexão à lista de conexões do programa e fica esperando a informação da CPN para verificar se é uma rede válida. A CPN então envia sua informação e a aplicação procura em todas as conexões que tivessem sido adicionadas anteriormente para avaliar se não existe nenhuma com o mesmo identificador. Se assim for, a conexão é aprovada pelo protocolo e fica pronta para iniciar transferências de dados. Em caso contrario, a conexão será rejeitada. No diagrama foram apresentadas a classe *Socket* e a *Socket pack* como duas classes diferentes, somente para dar uma visão clara do que esta dentro e fora da aplicação. Porém a classe *Socket pack* é herdada do *Socket*.

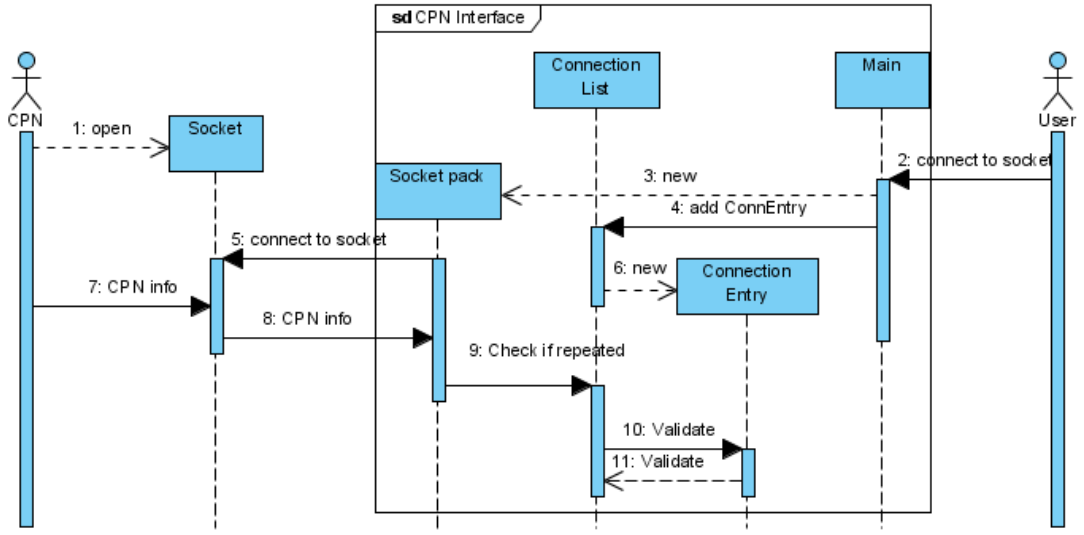


Figura 3.26 Diagrama de seqüência da conexão

3.2.5. Envio de dados a CPN Tools

O envio de dados a uma CPN pode ser realizado de forma ativa desde a aplicação. Depois que o usuário introduz os dados a serem enviados, o destino, e eventualmente adicionar uma origem (simulado) a escolha, a aplicação procura na lista das conexões a conexão já estabelecida com a rede de destino e envia a informação de acordo com o protocolo. No diagrama de seqüência da Figura 3.27, mostra-se no caso supondo que a conexão já foi estabelecida e validada.

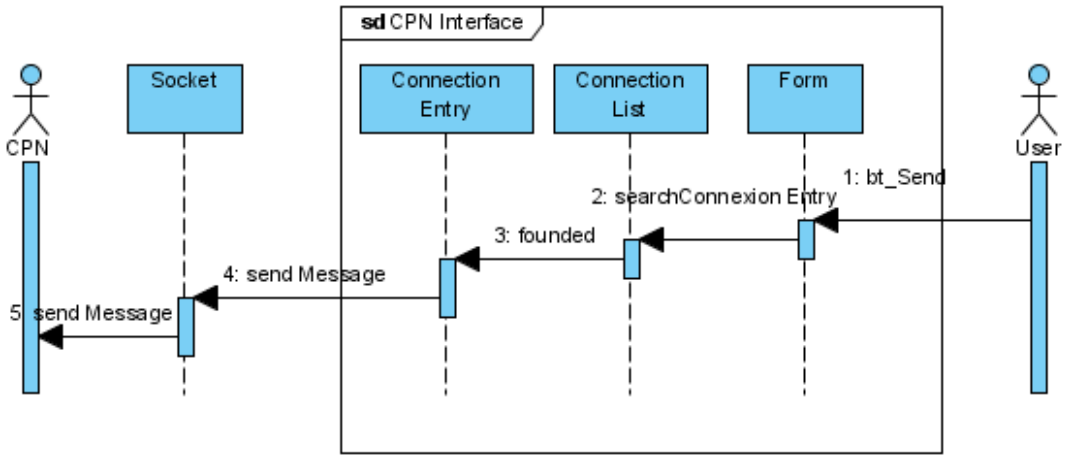


Figura 3.27 Diagrama de seqüência de envio de dados à CPN

3.2.6. Recepção de dados de CPN Tools

A Figura 3.28, mostra o diagrama de seqüência da recepção de dados na aplicação. Basta destacar que toda conexão ativada e avaliada encontra-se esperando dados assincronamente. O envio de dados da CPN executa a função *OnDataReceived* onde se manipula o pacote de acordo com o protocolo e se coloca na lista de espera (*QueueList*) da caixa de saída da conexão (o representado no diagrama) ou se encaminha para outros usos. Assim como o pacote de dados é destinado, volta-se a executar a função *WaitForData* esperando mais pacotes.

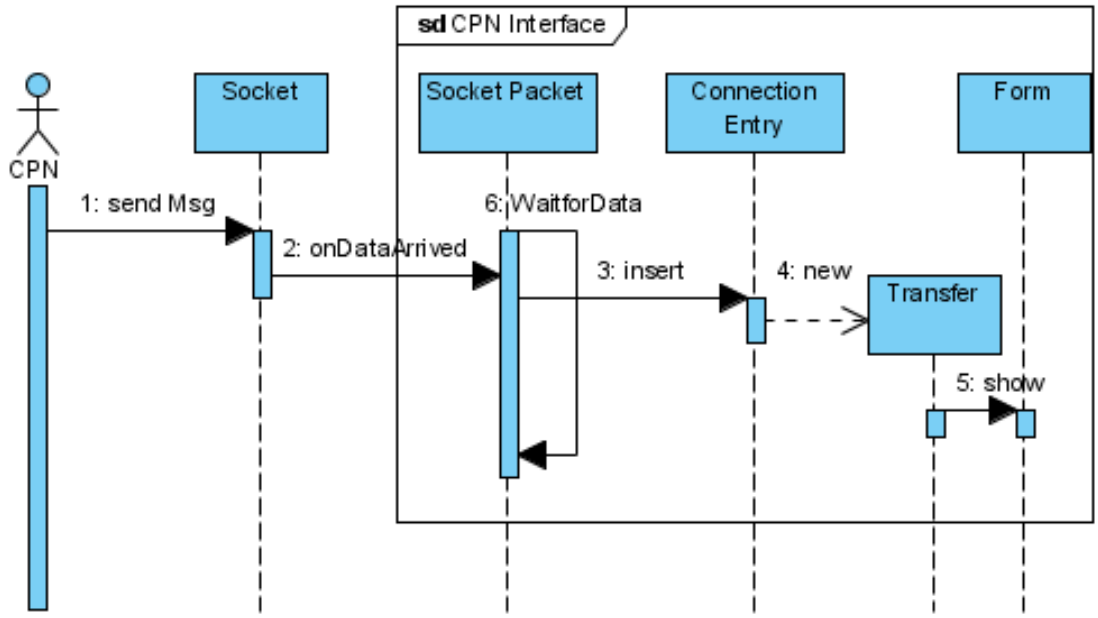


Figura 3.28 Diagrama de seqüência de recepção de dados desde a CPN

3.2.7. Comunicação entre várias CPN.

O seguinte diagrama de seqüência apresenta o funcionamento da comunicação que mais interessa, já que a aplicação está atuando como interface de comunicação entre duas CPN. A mensagem enviada é adicionada na caixa de saída e uma vez nela (seja por estar esperando mais pacotes que componham a mensagem ou pelo fato da rede de destino não se encontrar pronta para receber a transferência) é transmitida através do *socket* que está ligado à rede de destino.

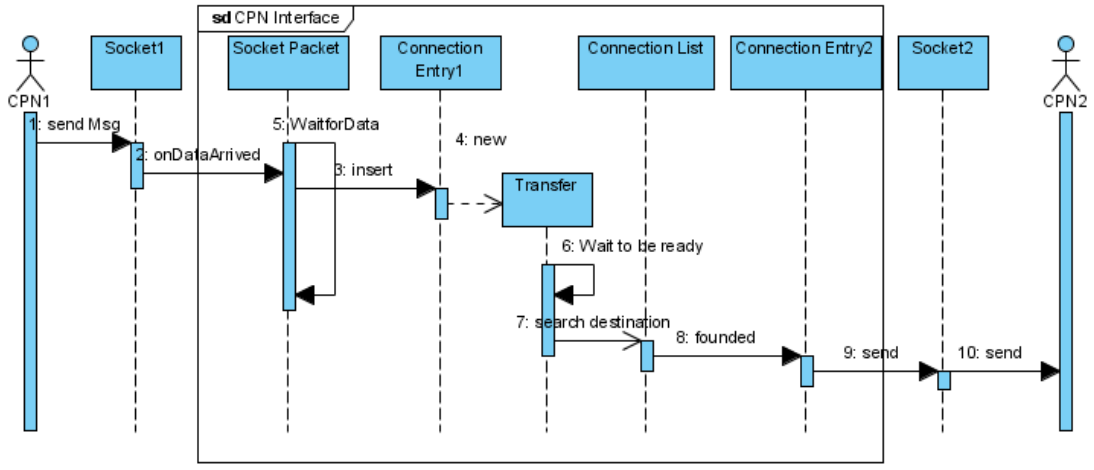


Figura 3.29 Diagrama de seqüência de comunicação entre duas CPN

3.3. Monitorar a simulação da CPN

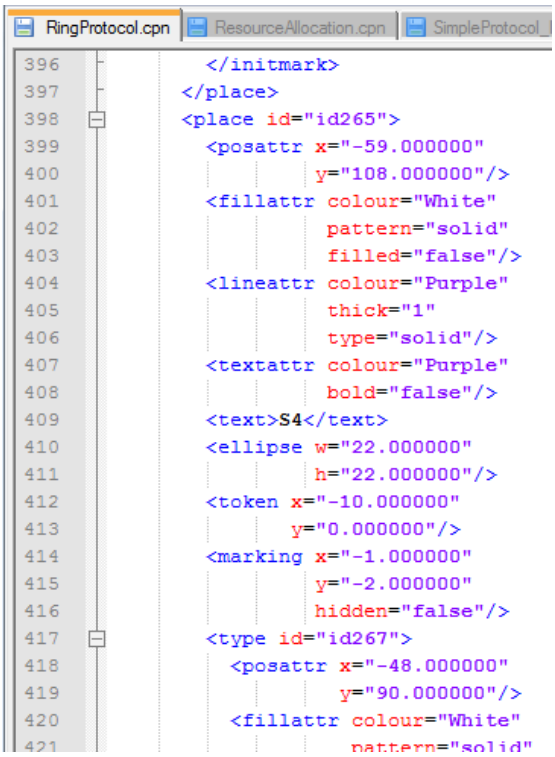
O monitoramento é um processo de comunicação igual aos anteriores, mas que possui um header particular para identificar a mensagem recebida e de monitoramento (caso do "Monitor" na Tabela 3.2). Além disso, o dado recebido, precisa ser decodificado e interpretado corretamente. Para que isso ocorra, é necessário primeiramente ter lido previamente o arquivo .cpn que vai ser monitorado, a fim de que o aplicativo conheça a arquitetura da rede a ser monitorada.

As características do dado a ser monitorado se definem a seguir:

- Recebe-se uma string que contém: números, o símbolo do dólar "$" e o "e comercial" ou *ampersand* "&".
- A *string* contém informação do número de *token*s que compõem todos os *places* da CPN, (sempre que foram escolhidos para ser monitorados) assim como o número identificador e único de cada *place*.
- O caráter "&"separa a *string* em informações referentes a diferentes *place*s e o "$" o identificador do número de *token*s.

3.3.1. Leitura do arquivo .cpn

A leitura do arquivo .cpn, onde o programa CPN Tools constrói a CPN é um passo necessário prévio para conseguir poder monitorar a rede, assim como para poder editá-la desde a aplicação desenvolvida. O arquivo da CPN tem uma estrutura interna igual à de um XML, unicamente com outra extensão de arquivo personalizada. Internamente lê-se o arquivo e vai se construindo uma classe com as distintas propriedades que contenha toda a informação útil do arquivo para poder trabalhar com esta classe internamente.



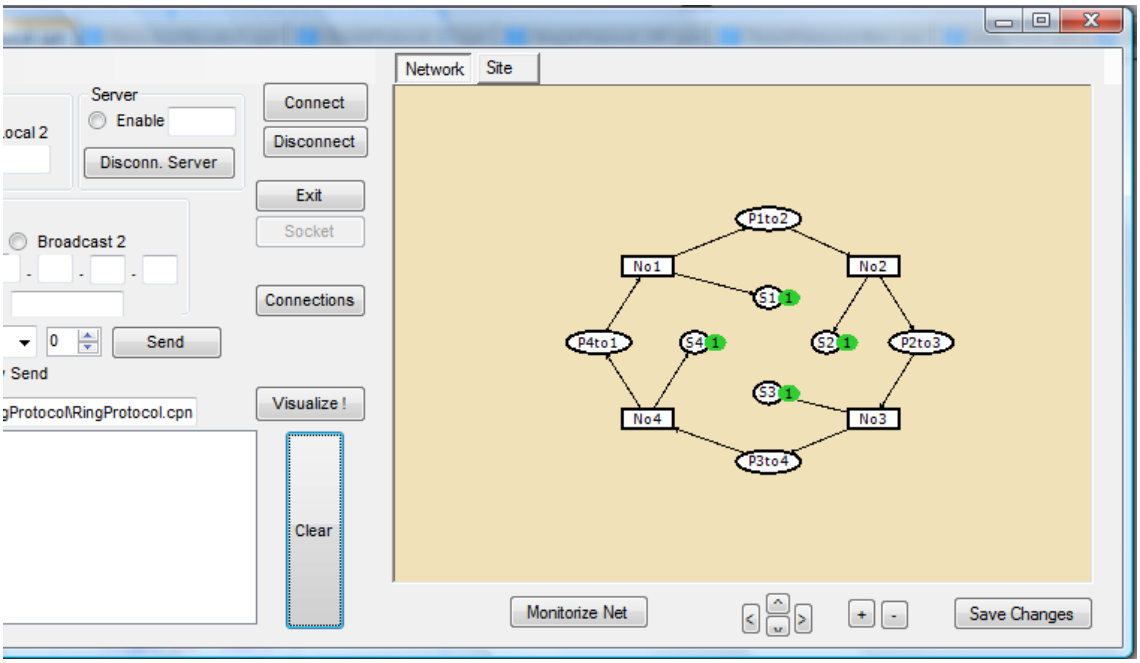


Figura 3.30 Construção de representação gráfica a partir do arquivo da CPN

3.3.2. Modificação automática do arquivo .cpn para ser monitorado

Depois de ter construído um objeto que contenha a informação relevante do arquivo da CPN, já é possível proceder com a edição do mesmo. Basicamente, o programa oferece dois tipos de modificações:

- Inclusão da biblioteca de comunicações desenvolvida neste projeto
- Edição da CPN para incluir instruções que desenvolvam comunicações e monitoramento.

3.4. Comunicação entre várias aplicações

A implementação do protocolo desenvolvido permite a comunicação, não só com o aplicativo CPN Tools, mas com qualquer aplicação externa, sempre que seja respeitado o protocolo desenvolvido. Porém, pode-se utilizar o protocolo TCP/IP simples com segmentação da mensagem (Raw Protocol), mas se perderá a capacidade de identificar a conexão, perdendo assim a possibilidade de transferir informação com o destino pré-determinado.

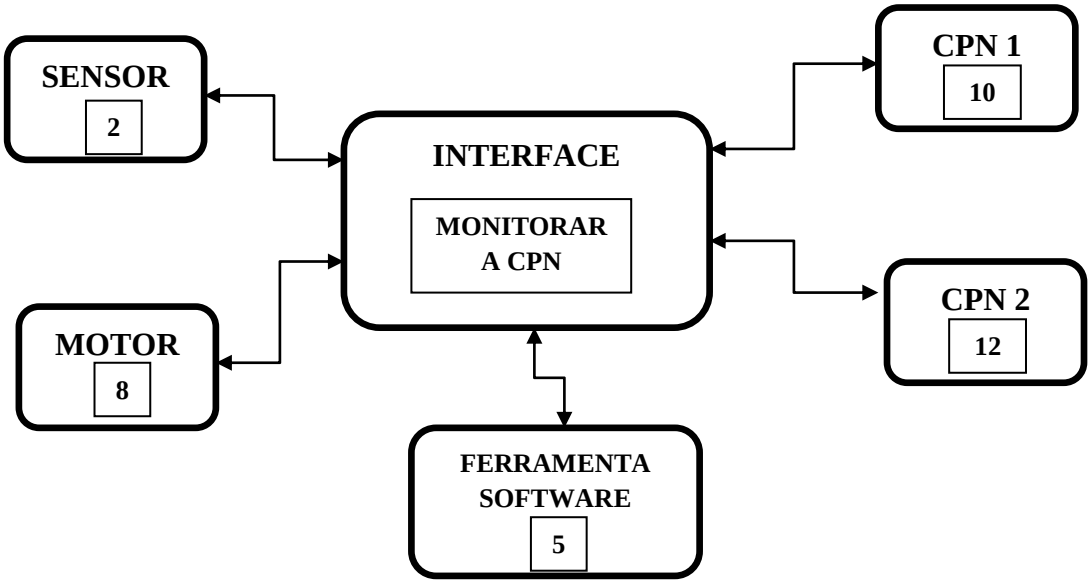


Figura 3.31 Arquitetura de um possível sistema a conectar mediante a Interface

Na Figura 3.31, mostra-se um exemplo aplicado de um sistema integrado por um bloco que possui um sensor, outro bloco com um controlador de motor, duas CPN e

uma ferramenta software. Por exemplo, segundo o decorrer da CPN, o motor estará ativado até que o sensor envie um sinal à CPN1. Esta, ao mudar de estado, manda um sinal à ferramenta para que o motor mude de velocidade. Para isso, outro sinal será enviado desde a ferramenta até o bloco controlador do motor. O número que aparece em cada bloco é o identificador numérico para as comunicações. Abaixo detalham- se as transações descritas acima (Não foram representados os cabeçalhos Header e Controle para maior simplicidade):

- Sinal desde o sensor(2) à CPN1 (10)

- Sinal desde a CPN1 (10) à ferramenta software(5)

- Sinal desde a ferramenta software(5) ao motor (8)

4. Teste de software

Como qualquer projeto de desenvolvimento de software, foi realizada uma série de provas para verificar o correto funcionamento do mesmo.

4.1. Testes das comunicações.

A seguir, mostram-se uma série de provas que demonstram o correto funcionamento das comunicações. Após a verificação correta do software em relação às comunicações foi realizada uma fase de testes de uma maneira convencional, para avaliar formalmente o correto funcionamento das comunicações. Nestas provas, provaram-se conexões do tipo "Rede A envia um dado à rede B gerida pela interface" ou para monitorar utilizando-se os dois protocolos desenvolvidos (Debug e Enabled). A seguir detalham-se estas provas:

- Prova A: As CPN utilizadas são similares às de envio e recepção já mostradas na Figura 3.7 e Figura 3.12, mas utilizam funções de codificação e decodificação apropriadas para a correta utilização do protocolo. Sendo esta a primeira das provas, foi feita em modo Debug, para observar lentamente o funcionamento.
- Prova B: Similar à prova A, porém esta vez é o servidor quem envia a informação, além de agir com formato de numeração das redes reduzido.
- Prova C: Aqui se testou o monitoramento, em modo *debug*, utilizando uma rede similar às dos filósofos (2)
- Prova D: Utilizando desta vez a CPN hierárquica que modela a célula de manufatura (Ver seção 5), com um número elevado de lugares monitorados e com carga de dados maior, testou-se em modo *Enabled* (automático) para certificar o correto funcionamento.

Arquivo	**Protocolo**	**Form. Estendido**	**Tamanho da Msg**
Prova A			
ReceiveProtocolServer	Debug	Estendido	>127 *byte*s
SendProtocolClient	Debug	Estendido	>127 *byte*s
Prova B			
SendProtocolServer	Debug	Reduzido	>127 *byte*s
ReceiveProtocolClient	Debug	Reduzido	>127 *byte*s
Prova C			
MonitorTestClient	Debug	Reduzido	<127 *byte*s
Prova D			
MonitorTestServer	Enabled	Estendido	<127 *byte*s
Prova E			
rede7k	Enabled	Reduzido	>127 *byte*s
Prova F			
pratosFil	Enabled	Reduzido	<127*bytes*

Tabela 4.1 Detalhe dos arquivos para os testes

Além de estas provas para comprovar o correto funcionamento das conexões, foram feitas as seguintes provas para averiguar o desempenho da interface em relação a o tempo requerido para processar os dados.

Prova E

Monitoramento de dados de 199-201 bytes por envio (dois blocos) obteve-se os seguintes resultados contando desde antes de começar a processar o primeiro bloco recebido até após ter processado o segundo:

	Teste 1	Teste 2	Teste 3	Global
Tempo médio de processamento (milisegundos)	26,29	30,11	27,47	27,91 msg
Núm de dados transferidos:	67	68	68	

Tabela 4.2 Tempos de processamento de dados para a prova E

Prova F:

Similar à anterior, mas com volume de dados menor, com blocos de 10 a 16 bytes.

	Teste 1	Teste 2	Teste 3	Global
Tempo médio de processamento (milisegundos)	6,62	6,07	6,77	6,48 msg.
Núm de dados transferidos:	52	48	116	

Tabela 4.3 Tempos de processamento de dados para a prova F

4.2. Análise dos resultados

Conclusivamente, obtiveram-se resultados satisfatórios, após porém de numerosas provas com pior resultado, mas que serviram para detectar falhas no desenho e na concepção da comunicação. Exemplos demonstrativos são, por exemplo, quando se enviou pela primeira vez uma mensagem com tamanho maior a 127 *byte*s, ou quando se tentou monitorar pela primeira vez uma CPN hierárquica.

Em relação aos tempos de processamento, são curtos e dependem em do volume de dados a processar. Além disso, as aplicações que vão ser utilizadas não requerem alto volume de dados, na maioria dos casos menor a um bloco (menor do que 127 bytes). No entanto, será necessário conhecer em qual aplicação vai se usar a interface para estabelecer uns critérios melhores para qualificar o desempenho da interface

5. Aplicações

A seguir mostram dois exemplos, onde a interface foi usada para se comunicar com uma CPN. Primeiramente se apresenta uma célula de manufatura e depois um exemplo simples de CPN conectado a uma ferramenta de representação gráfica.

5.1. Célula de manufatura

A CPN Tools em conjunto com a aplicação desenvolvida nesse projeto permitem o desenvolvimento de diversas aplicações, porém, a aplicação em célula de manufatura é a motivação ou o grande incentivo deste projeto.

A preparação do arquivo .cpn para o monitoramento deste tipo de redes hierárquicas compreende um grau superior de dificuldade. Alguns lugares e transições estão replicados, já que pertencem às redes principais e às subredes ao mesmo tempo, e deve-se evitar a possível redundância. A seguir mostra-se uma versão inicial da CPN hierárquica que modela a célula.

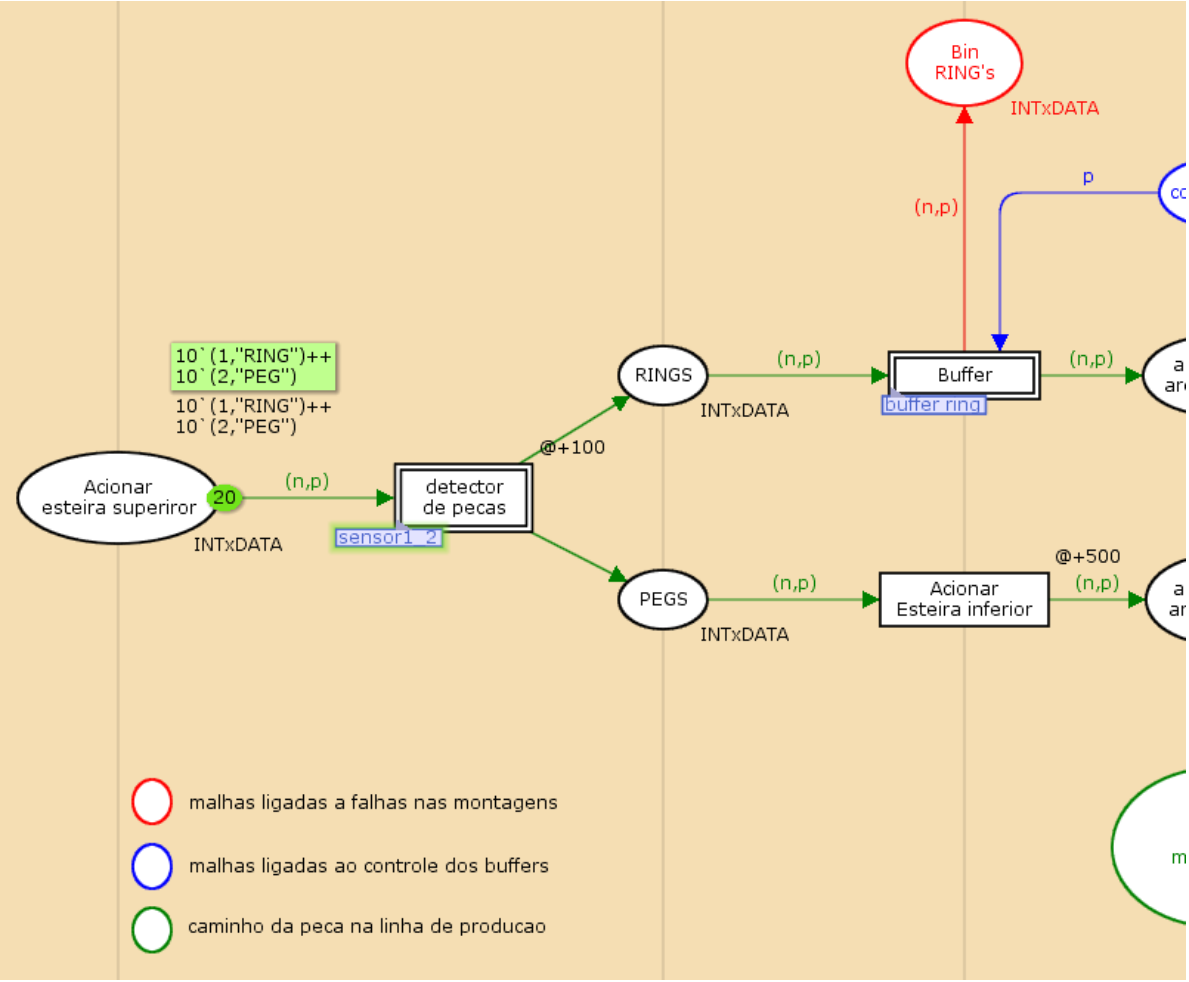


Figura 5.1 CPN principal da rede de manufatura (lado esquerdo)

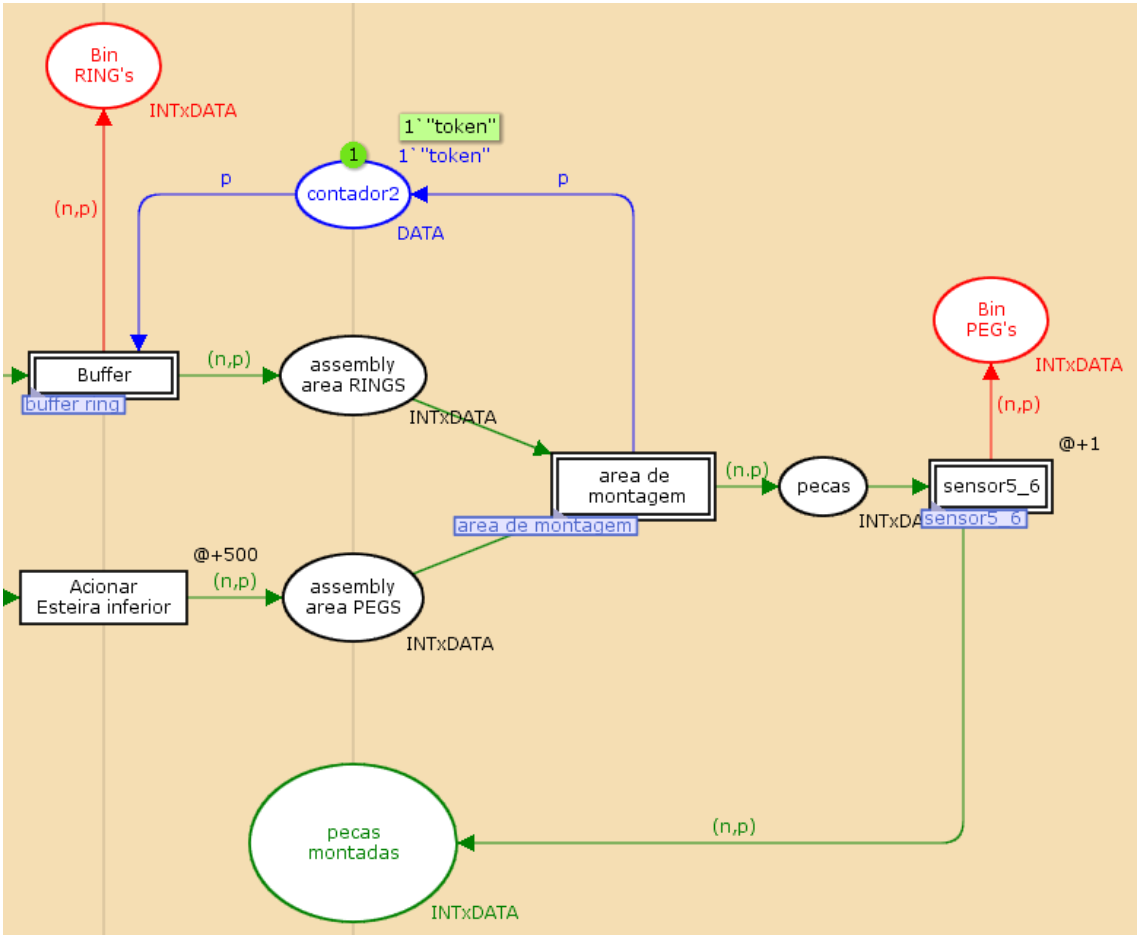


Figura 5.2 CPN principal da rede de manufatura (lado direito)

A rede possui quatro subredes, sendo elas dois sensores, uma área de montagem e um *buffer ring*.

Buffer ring:

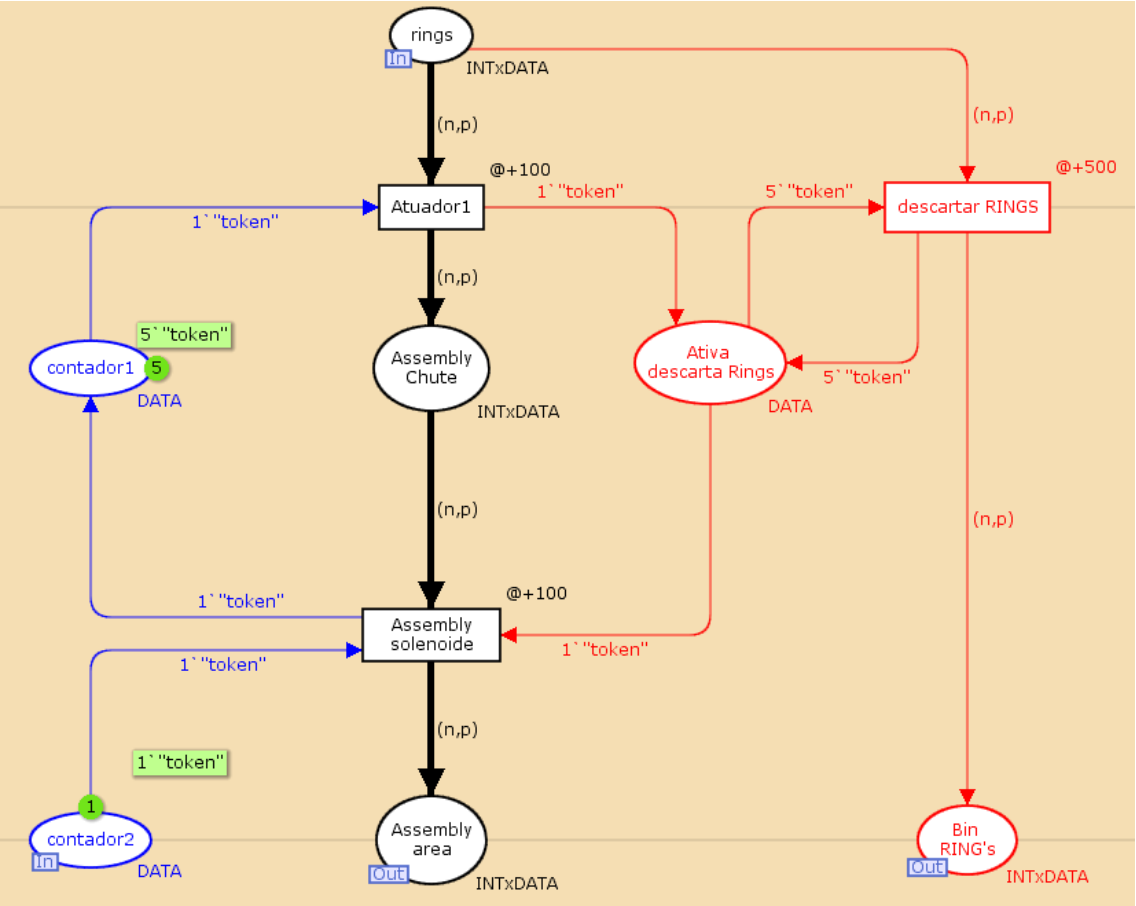


Figura 5.3 Subrede CPN do *buffer ring* da rede de manufatura

Bloco responsavel por:

Linhas azuis
1)Limitar o numero de "rings" no "Assembly Chute" atraves do contador1(max=5), que faz o papel de arco inibidor

2) Limitar o numero de "rings" na "Assembly area" para no maximo 1, fazendo uso do anti-lugar, "contador2"

Linhas vermelhas
1) quando o "assembly Chute" esta cheio, o lugar "Ativa descarta Rings" possue 5 tokens, ativando a transicao "descartar RINGS"

Figura 5.4 Descrição do *buffer ring* da rede de manufatura

Área de montagem:

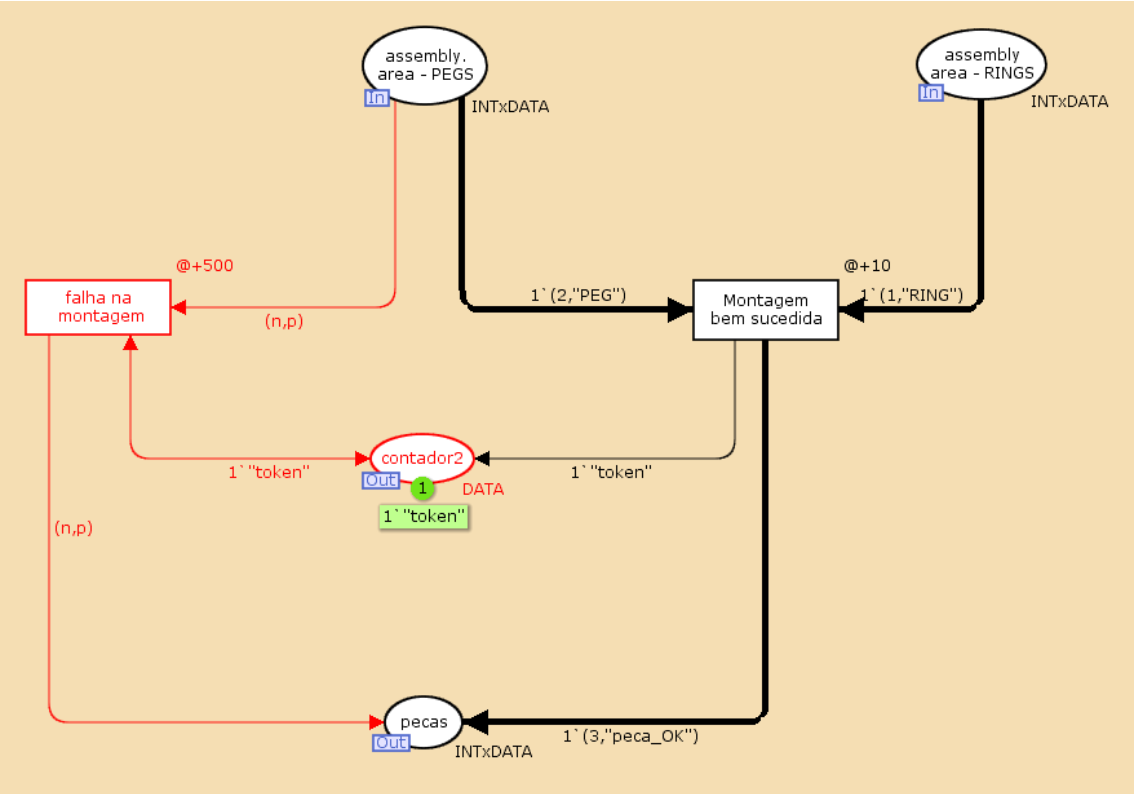


Figura 5.5 Subrede CPN da área de montagem rede de manufatura

Sempre que existirem PEG's e RING's disponiveis monta a peca, criando a variavel "peca_ok",
caso contrario ocorre falha na montagem (linhas vermelhas), e encaminha o PEG a frente

Figura 5.6 Descrição da área da rede de manufatura

Sensor 1/2

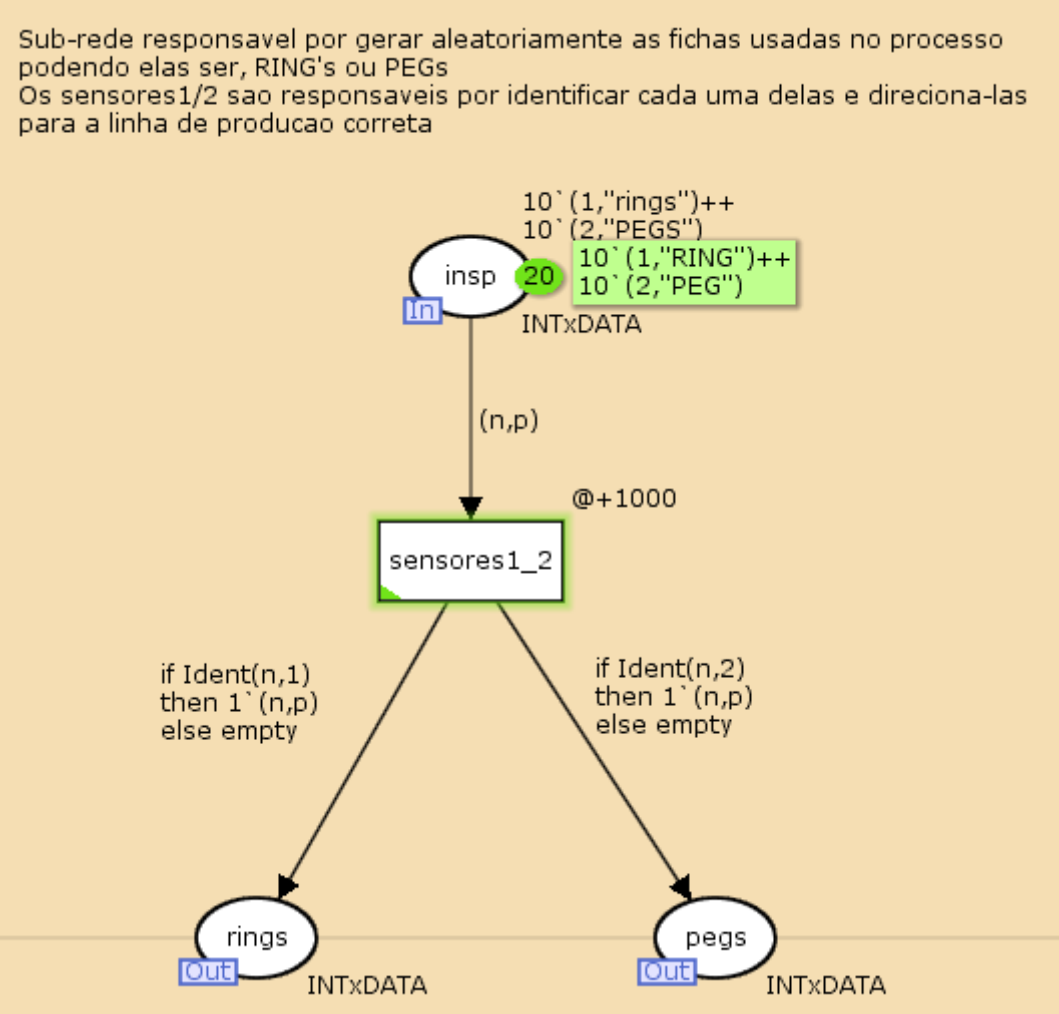


Figura 5.7 Subrede CPN do sensor 1/2

Sensor 5/6:

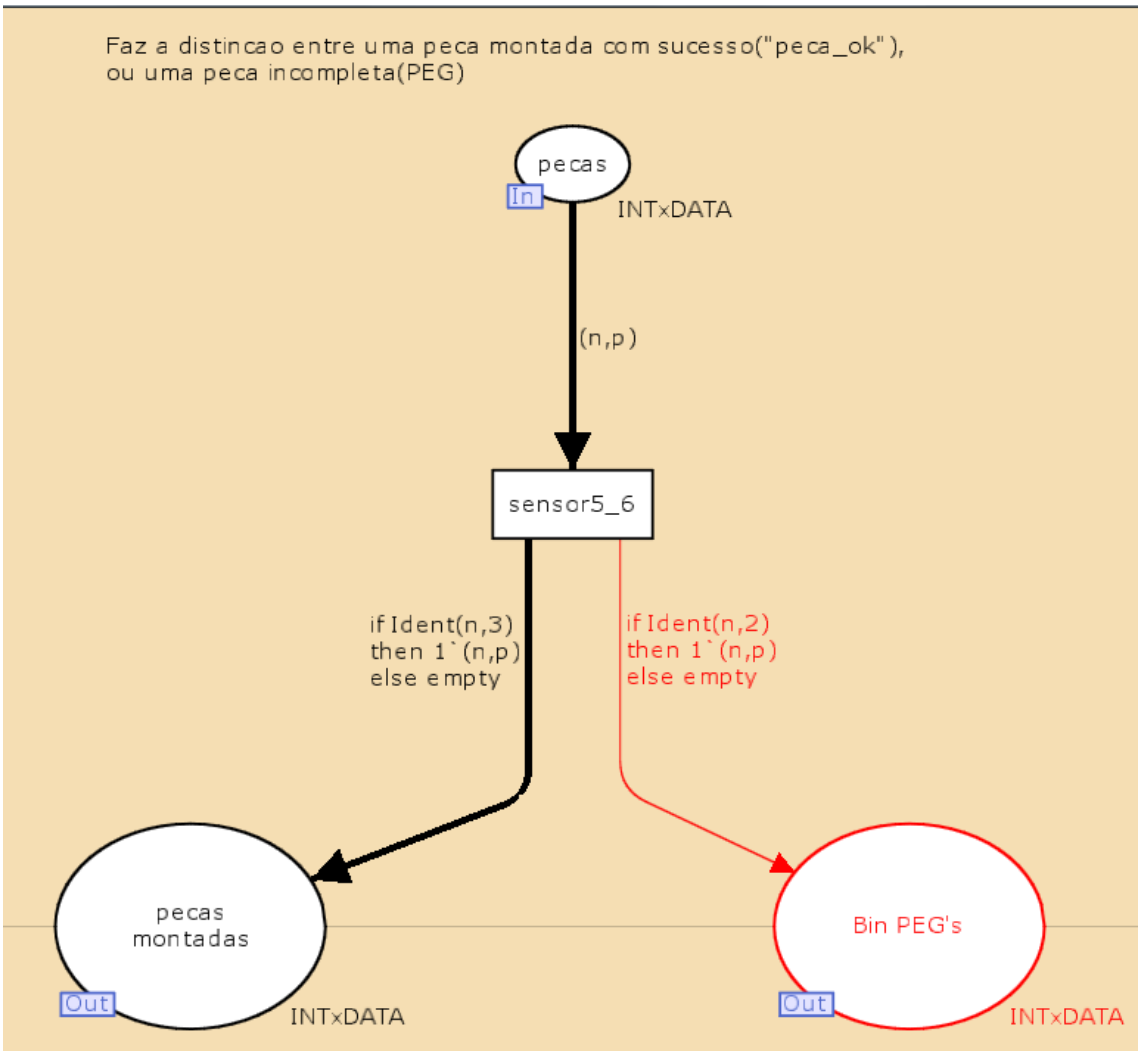


Figura 5.8 Subrede CPN de um dos sensores da rede de manufatura

5.2. CPN conectada a uma interface gráfica

Conectando uma CPN com uma interface gráfica externa, através da interface de comunicações, consegue-se provar a comunicação da ferramenta como elemento encaminhador dos dados transferidos.

Utilizando a mesma CPN já mostrada na Figura 2.4, elaborou-se um ambiente gráfico que representa os estados da CPN de maneira visual.

Figura 5.9 Software externo de representação gráfica da CPN da república. Estado inicial

Figura 5.10 Software externo de representação gráfica da CPN da república. Estado intermediário

6. Gestão do projeto

6.1. Seguimento cronológico

A seguir mostram-se as diferentes etapas que foram desenvolvidas ao longo do projeto, assim como acontecimentos importantes que direcionaram o decorrer do mesmo.

Etapas:

1. Definição do Trabalho
2. Estudo das CPN
3. Análise de viabilidade
4. Comunicação
   - 4.1. Comunicação simples como cliente
   - 4.2. Comunicação múltipla como cliente
   - 4.3. Comunicação simples como servidor
   - 4.4. Envio de dados
   - 4.5. Comunicação remota
   - 4.6. Implementação do protocolo próprio
   - 4.7. Comunicação múltipla completa
5. Monitoramento:
   - 5.1. Monitoramento estático (representação gráfica da CPN a partir do arquivo)
   - 5.2. Monitoramento dinâmico
   - 5.3. Monitoramento dinâmico de CPN hierárquica
6. Construção da biblioteca de comunicação
7. Inclusão automática da biblioteca de comunicação nos arquivos cpn
8. Edição das instruções de comunicação desde a aplicação
9. Documentação

Eventualidades:

A. Ampliação do projeto (protocolo e edição dos arquivos cpn desde a aplicação)
B. Reavaliação da utilização da comunicação com TCP/IP (*stream* de dados em lugar de pacotes de dados)

Mês	Sep	Out	Nov	Dec	Jan	Feb	Mar	Abr	Mai	Jun	Jul	Ago
Etapas												
1	X											
2	X	X	X									
3		X										
4.1		X	X	X								
4.2				X	X	X						
4.3						X	X					
4.4							X	X				
4.5						X	X	X				
4.6							X	X	X	X		
4.7									X	X	X	
5.1				X	X	X						
5.2								X	X			
5.3									X	X		
6								X	X	X		
7									X	X		
8										X	X	
9							X	X	X	X	X	X
A						X						
B							X					

Tabela 6.1 Cronograma do projeto

6.2. Ferramentas utilizadas

As ferramentas utilizadas foram escolhidas acomodando-se as já disponíveis ou as indicadas pelo CRTi. As mesmas estão listadas a seguir:

Desenvolvimento:

- CPN Tools: Ferramenta principal de modelagem e análise de CPN.
- Visual Studio 2005: Ferramenta preferencialmente utilizada no CRTi para desenvolvimento de software
- Standard ML of New Jersey 110.0.7: Ferramenta de desenvolvimento em SML para facilitar o desenvolvimento das bibliotecas de comunicação incorporadas depois em CPN Tools
- Notepad++: Ferramenta de edição de arquivos de diversa extensão que facilita a pesquisa e a edição dos mesmos
- Compare it!: Aplicação para a comparação de arquivos, especialmente útil na manipulação de arquivos XML.

Documentação

- Microsoft Word 2007: Ferramenta editora de textos.
- Visual Paradigm for UML: Ferramenta para construção de diagramas UML

7. Conclusões e perspectivas futuras

Este projeto foi desenvolvido com a finalidade de estabelecer a capacidade de comunicação entre diferentes módulos que formam parte de um sistema que seja modelado por uma CPN. Desta maneira, pode-se analisá-lo, monitorar o andamento, e até mesmo, remotamente, atuar no mesmo. Enfim, aparecem numerosas possibilidades. Portanto este projeto e seu aplicativo têm um espírito integrador, para poder unir distintos elementos previamente desenvolvidos e outros que ainda nem foram concebidos.

Em relação aos aspectos da comunicação, o aprofundamento para conhecer a dinâmica do protocolo TCP/IP foi muito produtivo, não unicamente para aprender suas características e propriedades, mas também para solucionar alguns problemas surgidos durante o desenvolvimento do projeto. Dificuldades estas que normalmente são ignoradas no dia a dia apesar de que este protocolo é usado por milhões de pessoas diariamente. A solução implementada, em relação à capacidade comunicativa, tem a intenção de oferecer uma solução genérica com o menor número de limitações possíveis por parte da aplicação. A intenção consiste em conseguir adaptar-se aos elementos com os quais estabelece comunicação e não que estes precisem adaptar-se à aplicação. Assim, o protocolo desenvolvido proporciona espaço para incluir novas informações sem alterar substancialmente o protocolo nem os cabeçalhos.

Contudo, ao decorrer do projeto, apareceram novos requisitos, e teve-se que conceber várias idéias anteriormente trabalhadas e implementar outras novas. Este fato é bem freqüente tanto no mundo empresarial como acadêmico, mostrando assim, a relevância e a complexidade do mesmo.

Durante o desenvolvimento do projeto, foram tratados diversos tópicos do mundo da engenharia, desde a geometria analítica para representar a CPN graficamente, até a mineração de dados para unicamente escolher os dados necessários do arquivo da CPN, passando obviamente pelos conceitos de comunicação através de TCP/IP.

As características da aplicação superaram amplamente os objetivos inicialmente fixados, facilitando o trabalho posterior feito pelo CRTi, incluindo facilidades como a capacidade de monitorar uma CPN remotamente, facilitar a edição da CPN desde a aplicação desenvolvida e a incorporação automática da biblioteca de comunicação nos arquivos que definem a CPN.

Após a finalização deste projeto, deve ser feita a efetiva integração da CPN Tools com outras aplicações ou elementos físicos como sensores ou outros componentes.

Com relação ao aspecto comunicativo da aplicação desenvolvida, as linhas de trabalho futuras podem compreender, por exemplo, acrescentar novos canais de envio da informação como, por exemplo, Bluetooth ou Zigbee, a preparando-se para mudanças no formato de comunicação como, a transição dos endereços IP, de IPv4 a IPv6. Devido a uma questão de prioridades, foi abandonado o desenvolvimento de um ambiente gráfico que representasse os estados de uma CPN com imagens em troca de uma maior editabilidade automática dos arquivos cpn desde a aplicação desenvolvida. Este ambiente gráfico poderia ser então posteriormente desenvolvido.

8. Referências bibliográficas

1. Wünschmann, S. ***Identitätbasierte Kryptographie für sicheres Workflow-Management.*** **Institut für Wirtschaftsinformatik, Universität Regensburg. Regensburg : s.n., 2007. Dissertação de mestrado.**

2. Group, CPN. **Comms/CPN functions.** ***Functions for TCP/IP communication between the simulator and external processes.*** **[Online] http://wiki.daimi.au.dk/cpntools-help/commscpn_functions.wiki?cmd=get&anchor=Comms/CPN+functions.**

3. Petri, C. A. ***Kommunikation mit Automanten.*** **Rheinisch-Westfälischen Institues für Instrumentelle Mathematik, Universität Bonn. 1962. Tese de Doutorado.**

4. ***Carl Adam Petri and "Petri Nets".*** Reisig, W. **3, 2009, Fundamental Concepts in Computer Science, pp. 129-139.**

5. Holt, A. W. ***Final Report of the Information System Theory Project.*** **Rome Air Development Center, Griffiss Air Force Base. New York : s.n., 1968. RADC-TR-68-305.**

6. Holt, A. W. e Commoner, F. ***Events and Conditions.*** **Applied Data Research, Massachusetts Computer Associates. New York : ACM, 1970. Technical Report.**

7. Reising, W. ***Petrinetze - Eine Einführung 2, überarbeitete und erweiterte Auflage.*** **Berlin : Springer, 1986.**

8. ***Petri Nets: Properties, analysis and applications.*** Murata, T. **77 (4), 1989, Proc. of the IEEE, pp. 541-580.**

9. Jensen, Kurt and Rozenberg Grzegorz. ***High-level Petri nets: theory and application.*** **London : Springer-Verlag, 1991. ISBN 3-540-54125-X.**

10. ***Special Section on The Practical Use of High-Level Petri Nets: Preface by the section editor.*** Jensen, K. **(4), 2001, STTT, Vol. 3, pp. 269-371.**

11. Jensen, K. ***Net Modeller i Systembeskrivelse.*** **Århus Universitet. s.l. : Århus Universitetsforlag, 1980. Summary of Ph. D. thesis.**

12. Genrich, H. and Lautenbach, K. **The Analysis of Distributed Systems by Means of Predicate / Transition-Nets.** ***Semantics of Concurrent Computation.*** **1979, pp. 123-147.**

13. ***Coloured petri nets and the invariant-method.*** Jensen, K. **3, 1981, Theoretical Computer Science, Vol. 14, pp. 317-336.**

14. ***Introdução às Redes de Petri e Aplicações.*** Maciel, P.R.M., Lins, R.D. e Cunha, P.R.F. **Campinas : s.n., 1996. 10ª Escola de Computação.**

15. Jensen, K e Kristensen, L.M. ***Coloured Petri Nets. Modelling and Validation of Concurrent Systems.*** **Berlin : Springer-Verlag, 2009.**

16. Huber, P, Jensen, K e Shapiro, R.M. **Hierarchies in coloured petri nets.** ***Lecture Notes in Computer Science.*** **s.l. : Springer Berlin / Heidelberg, 1991, pp. 313-341.**

17. Beaudouin-Lafon, M., et al. **CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets.** ***Coloured Petri Nets, Application and Theory of Petri Nets 2001, Proceedings of the 22nd International Conference, ICATPN 2001 Newcastle upon Tyne.*** **s.l. : Springer-Verlag, 2000, pp. 71-80.**

18. CPN Group, Aarhus University. **CPN Tools License Agreement. [Online] 2004. http://wiki.daimi.au.dk/cpntools-help/license_agreement.wiki.**

19. ***Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN.*** Gallasch, G e Kristensen, L. M. **Aarhus : s.n., 2001. Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. pp. 75-91.**

A. Apêndice: Manual de usuário de CPN Communications Interface

Sumário

Requisitos e limitações
Aspecto visual. Primeiros passos.
    Instalação do programa
    Aspecto visual
Comunicação
    Interface como cliente, CPN Tools como servidor
    Interface como servidor, CPN Tools como cliente
Monitoramento
Inclusão da biblioteca e parâmetros de comunicação
Instruções de comunicação na CPN
Salvar as modificações da CPN
Tabela da rede para as provas.
Procedimentos incorretos previstos.
Conectar a Interface com o exterior
Construir uma função de (de)codificação
Arquivo SMLFunctions.xml

I. Requisitos e limitações

Sistema operativo	Provado em Windows XP, Vista e Windows7. Provado em Linux Ubuntu após ser compilado com Mono.
Num. de conexões	Identificáveis: 256 elementos diferentes (de 0 a 255).
String identificadora da conexão	Tamanho máximo: 124 caráteres.
Places	Todos os *places* têm que ter um nome associado.
Arcos	Todos os arcos têm que ter uma expressão que defina o peso de arco.
Nomes na CPN	Os nomes das transições, os *places*, e as subredes hierárquicas unicamente podem ter os seguintes caráteres: alfanuméricos, underline (“_”) e espaço (“ ”).

Tabela A.1 Requisitos e limitações

II. Aspecto visual. Primeiros passos.

Instalação do programa

A instalação é simples e consta de três passos:

I. Abrir a pasta de instalação do aplicativo CPN Tools Connection Interface e executar setup.exe.

II. O sistema verificará se todos os requisitos necessários estão disponíveis e pedirá a confirmação do usuário para proceder com a instalação.

Figura A.1 Verificação dos requisitos durante a instalação

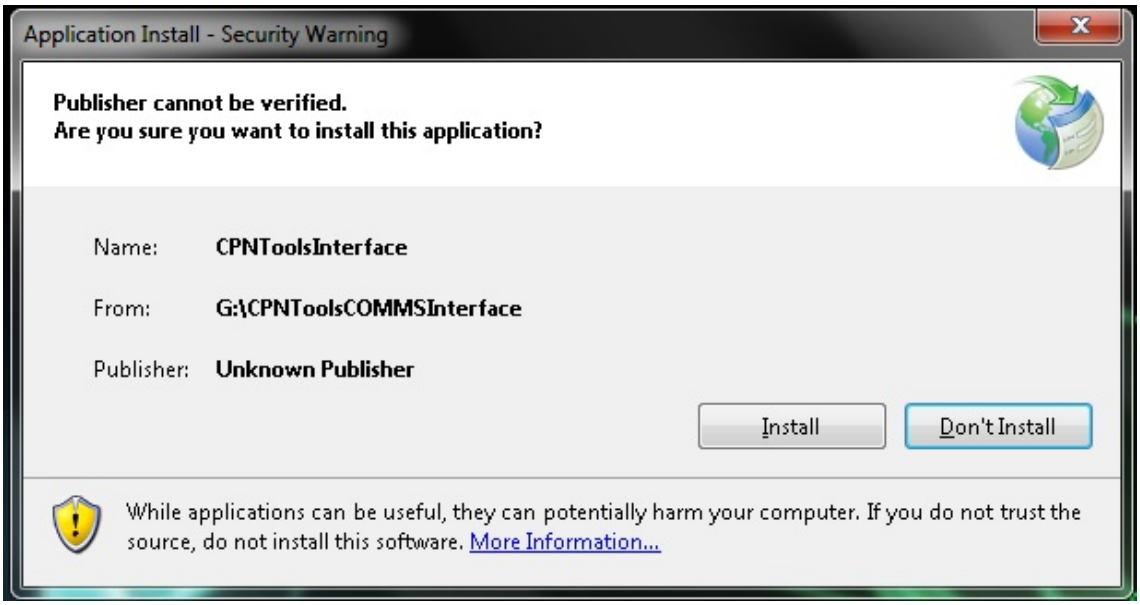


Figura A.2 Confirmação da instalação

III. Após aceitar, o programa abrirá automaticamente, havendo-se criando um no menu de inicio.

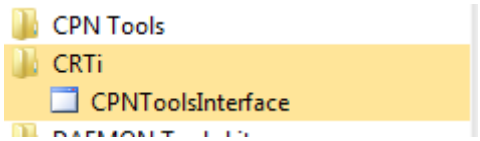


Figura A.3 Aspecto da instalação no menu de inicio.

Aspecto visual

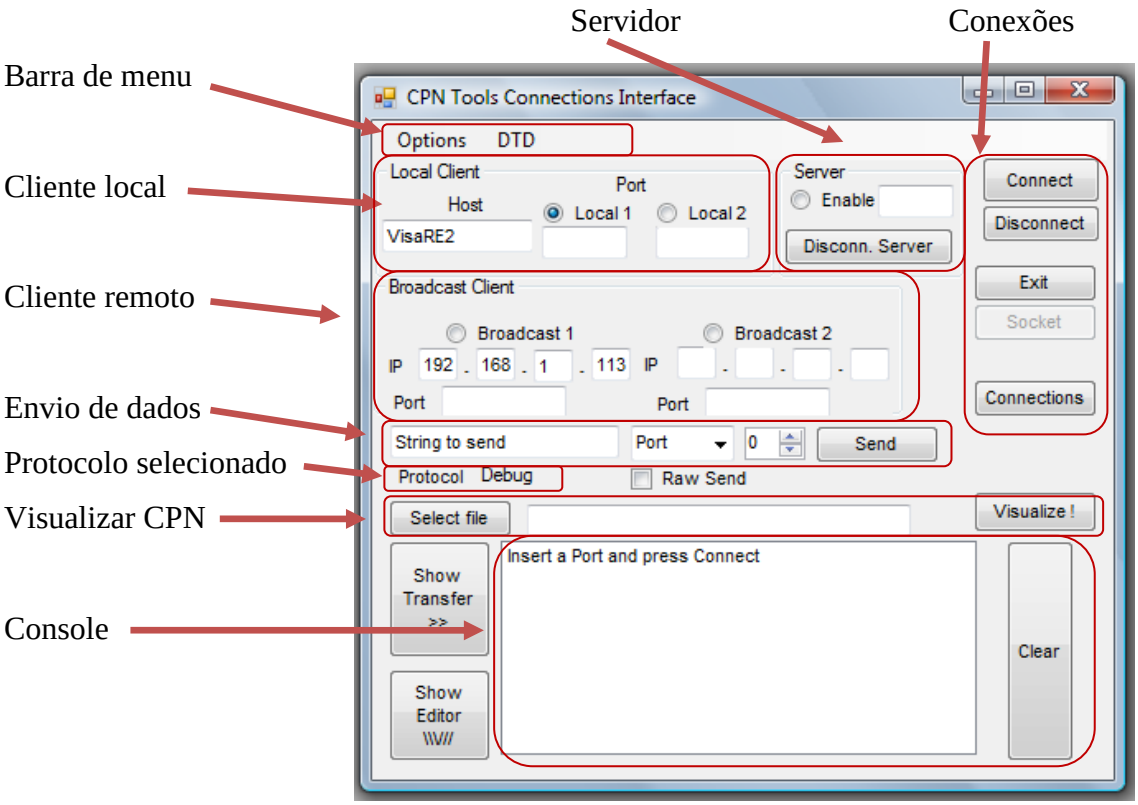


Figura A.4 Aspecto inicial da aplicação.

- Barra de menu: Aqui pode-se mudar o protocolo entre outras opções
- Clientes locais e remotos: Para conectar-se como cliente a uma porta e a uma direção IP (unicamente no caso de remoto). Embora só ter 2 campos, aceita até 256 conexões.
- Envio de dados: Para enviar dados comandado pelo aplicativo à uma conexão aberta. O seletor numérico indica que número de conexão teria o programa ao enviar a *string* simulando uma conexão já aberta.

- Visualizar CPN: Para visualizar a CPN, o que permite o monitoramento e a edição de instruções de conexão.
- Console: Onde o programa mostra suas mensagens.
- Conexões: Botões e indicadores para abrir, encerrar e acompanhar as conexões existentes.

II. Comunicação

Existem três protocolos para serem utilizados.

- Protocolo "Raw": É o protocolo utilizado pela CPN Tools, TCP/IP enviando unicamente o cabeçalho com o tamanho do pacote enviado.
- Protocolo "Enabled": Adiciona um *byte* de controle, e outro de origem e destino.
- Protocolo "Debug": É uma versão idêntica a Enabled, no entanto paralisa as transações para poder segui-las ativando elas passo a passo.

A continuação mostra-se como realizar conexões simples utilizando os arquivos de exemplo fornecidos pelo programa.

1. Interface como cliente, CPN Tools como servidor

Neste caso serão utilizados somente os arquivos com extensão cpn cujo nome termina com "Server". O servidor é sempre quem inicia a comunicação, então o primeiro passo deve ser efetuado na rede CPN abrindo o servidor e depois, desde a interface conectando-se como cliente ao servidor.

1º. Iniciar a simulação no CPN Tools

Figura A.5 Ferramentas de simulação de CPN Tools.

2º. Conectar ao servidor na Interface: Para isso clicar em "Local 1" ou "Local 2" (ou "Broadcast" também introduzindo as IP de destino) e preencher o número da porta que tem que ser o mesmo definido na conexão do servidor. (Ver as portas de cada arquivo .cpn). Clicar "Connect". Cabe salientar que ao clicar em qualquer dos *radiobuttons* do local, o programa autodetecta o IP local, informação que pode ser útil.

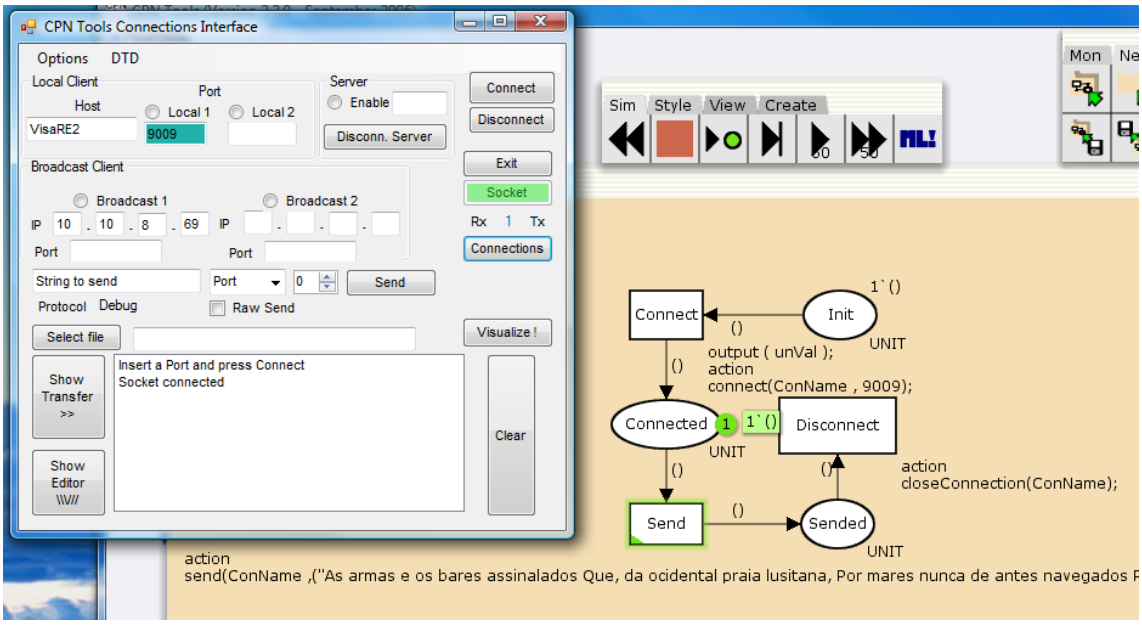


Figura A.6 Conexão entre a aplicação e CPN Tools.

3º. Se a conexão for bem sucedida, iluminar-se-á o botão "Socket" de verde e aparecerão uns indicadores embaixo dele: A *label* "Rx" ilumina-se em azul mostrando que foi recebido algum dado. O número que aparece representa o número de conexões abertas e a *label* "Tx" ilumina-se em vermelho quando se envia um dado para o exterior.

Figura A.7 Indicador do estado das conexões.

4º. O botão "Connections" mostra uma janela onde aparecem com mais detalhe as conexões abertas. Fazendo duplo clique em uma delas aparece uma lista dos pacotes

de dados prestes a serem enviados. Esta funcionalidade só é útil com o protocolo Debug ativado (já que com "Enabled" os envios são feitos de maneira automática).

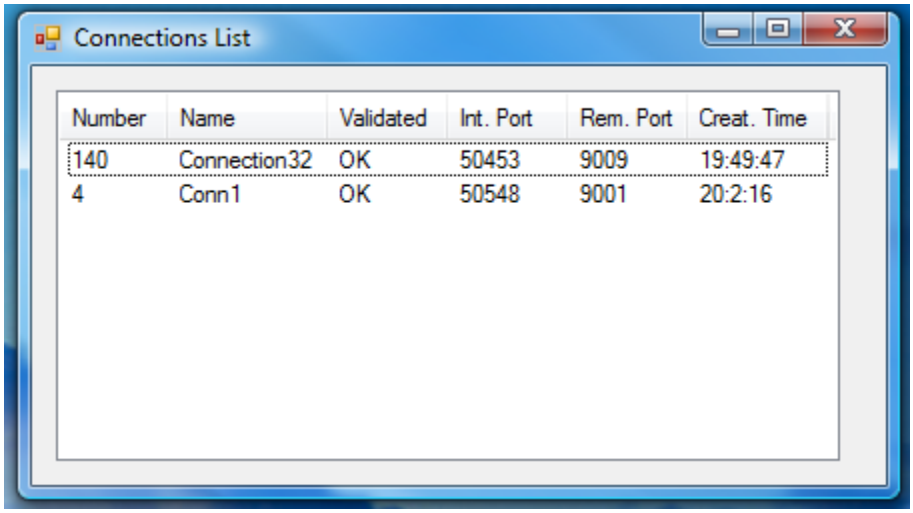


Figura A.8 Lista de conexões ativas.

5º. (Só válido para "Debug") Fazendo duplo clique ou selecionando "Open Queue List" após pressionar o botão direito na conexão que vai enviar um pacote de dados, aparecerá outra janela chamada "Queue List" onde aparecerão os pacotes prontos a serem enviados. Fazendo duplo clique em um deles, enviar-se-á o pacote só se estiver pronto para isso.

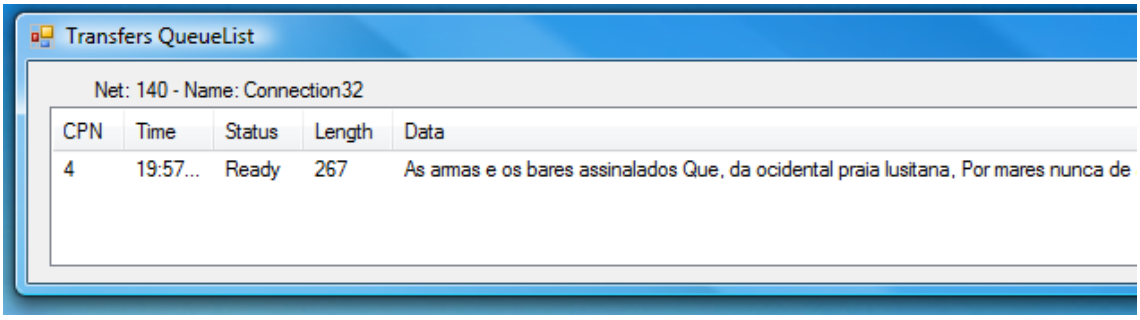


Figura A.9 Caixa de saída das transferências.

6º. Se quiser enviar qualquer dado desde a interface à CPN Tools, escrever o dado onde aparece "String to send", selecionar a conexão onde vai ser enviada, escolher o número de rede de origem e clicar "Send". Se for clicada "Raw Send" será enviado o dado sem informação de controle e de origem e destino (RAW protocol) mesmo que estiver ativado o protocolo.

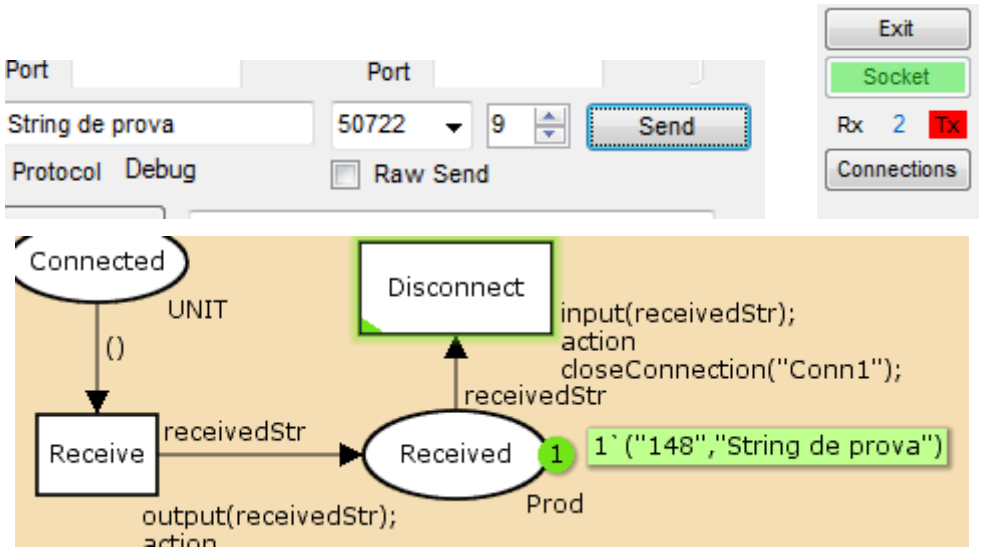


Figura A.10 Aspecto gráfico durante o envio de dados.

A Figura A.10 mostra um exemplo para comprovar o correto funcionamento onde se utiliza uma decodificação especial que mostra o cabeçalho *Src&Dst* sem separação (diferente da função *receiveDecode*, ver Figura 3.25). Neste caso o número 148, representa o cabeçalho Src&Dst (origem e destino). 148 em binário é 10010100. Como já foi descrito na Tabela 3.1, o número tem que ser partido em duas partes para uma correta interpretação. 1001 a origem e 0100 o destino. A origem foi escolhida ao enviar o dado (9), e o destino foi determinado pela porta escolhida (4) como se pode observar na figura seguinte. A porta 50722 coincide com a rede 4.

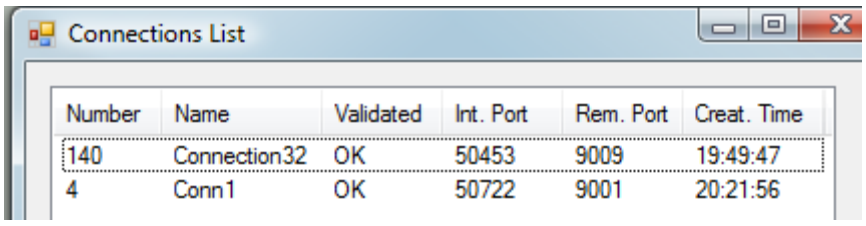


Figura A.11 Lista de conexões.

7º. Para desconectar todas as conexões abertas, pode se clicar no botão "Disconnect" e assim pode-se ver que o botão "Socket" irá se desligar.

2. Interface como servidor, CPN Tools como cliente

Neste caso serão utilizados somente os arquivos com extensão cpn cujo nome termina com "Client". O Servidor é sempre quem inicia a comunicação, então o primeiro passo deve ser efetuado na interface abrindo o servidor e depois na CPN Tools conectando como cliente ao servidor.

1º. Conectar ao servidor na Interface: Primeiramente clicar em "Server" e preencher o número da porta que precisa ser o mesmo definido na conexão do servidor. (Ver depois as portas de cada arquivo .cpn). Clicar "Connect". O servidor continuará esperando sucessivas conexões na mesma porta até fechar a interface.

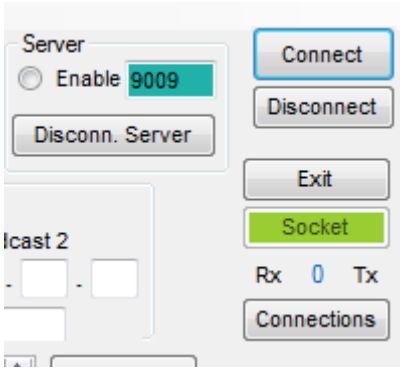


Figura A.12 Servidor aberto pela aplicação.

2º. Iniciar a simulação na CPN Tools

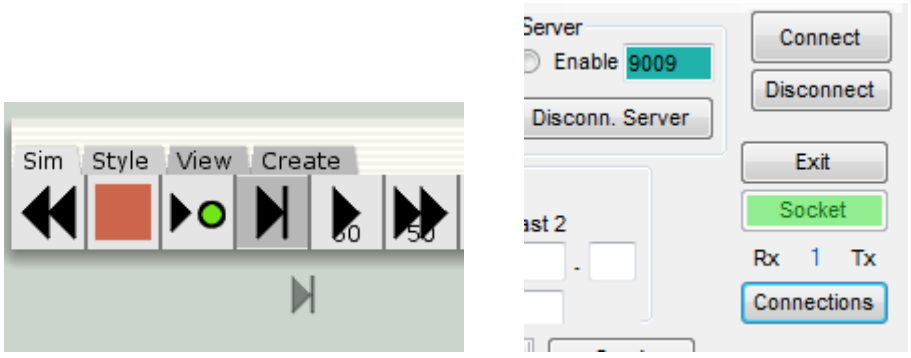


Figura A.13 Servidor aceitando conexão de entrada.

3º, 4º, 5º e 6º igual que no caso anterior.

7º. Neste caso a desconexão pode ser feita totalmente, fechando todas as conexões com o "Disconnect", ou parcialmente desconectando só as conexões que foram aceitas por uma determinada porta clicando em "Disconn. Server". Para isso deve-se indicar tal porta na *Textbox*, localizada acima deste botão. Cabe salientar que só serão desconectadas as conexões dessa porta selecionada, ficando operantes as demais portas. Também serão aceitas futuras conexões na porta "desconectada".

III. Monitoramento

Para monitorar a CPN, deve-se visualizar o arquivo (.cpn). Para isso, clicar em "Select file", selecionar a rede e clicar em "Visualize". Esta operação demora um pouco, já que faz a verificação se o arquivo CPN está em referência ao arquivo de extensão DTD (Document Type Definition), o qual se encontra na internet (www.daimi.au.dk/~cpntools/bin/DTD/5/cpn.dtd). Sem conexão com a internet, ocorrerá erro na aplicação. Pode-se substituí-la de forma manual, já que este arquivo encontra-se na pasta "C:\Program Files\CPN Tools\converter\cpn.dtd". Se a leitura do DTD for bem sucedida, a janela da aplicação mudará de tamanho e mostrará a rede. Pode-se mover a rede arrastando o mouse ou com os botões representados por setas abaixo da visualização da CPN. Também se pode dar zoom com "+" e "-". Uma vez desenhada a rede, pode-se começar a monitorá-la sem problemas. As redes preparadas para o monitoramento começam por "Monitor"

Para que uma CPN possa ser monitorada dinamicamente, devem ser incluídas as bibliotecas de comunicação assim como um monitor que envie as informações dos *tokens*, cada vez que uma transição for disparada.

IV. Inclusão da biblioteca e parâmetros de comunicação

Para aceder nesta função, uma vez visualizada a CPN, clicar no botão "Monitorize Net". Aparecerá uma janela como a seguinte:

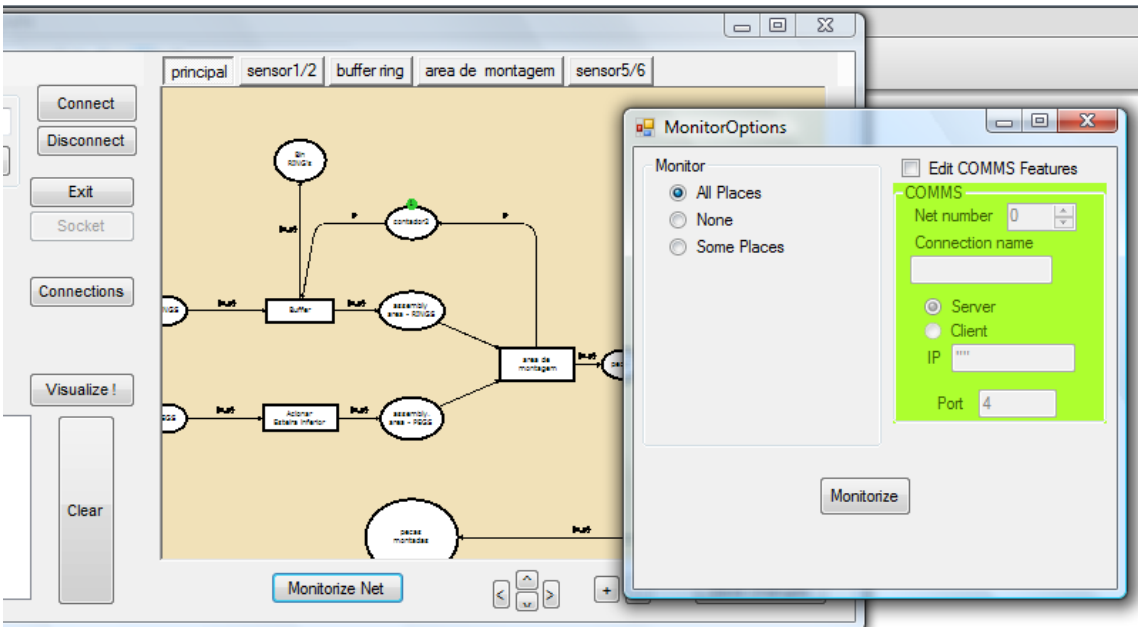


Figura A.14 Edição dos parâmetros da comunicação.

No lado esquerdo mostram-se botões para o monitoramento dinâmico, onde podem ser escolhidos todos os *places*, nenhum ou alguns. A direita aparecem os parâmetros de comunicação da rede, onde pode-se introduzir o número da rede, a *string* identificadora da conexão, definir a conexão como cliente ou servidor, o IP e a porta. Neste caso aparece com a cor verde, o que indica que estes parâmetros já foram introduzidos anteriormente e que possuí a biblioteca de comunicações. Naturalmente podem ser editados nesta janela clicando na opção para editá-los. Por outro lado, a figura seguinte mostra uma rede sem biblioteca nem parâmetros de comunicações. Por isso aparecerá sem a cor verde, e a opção anterior de editá-los e trocada pela de adicioná-los.

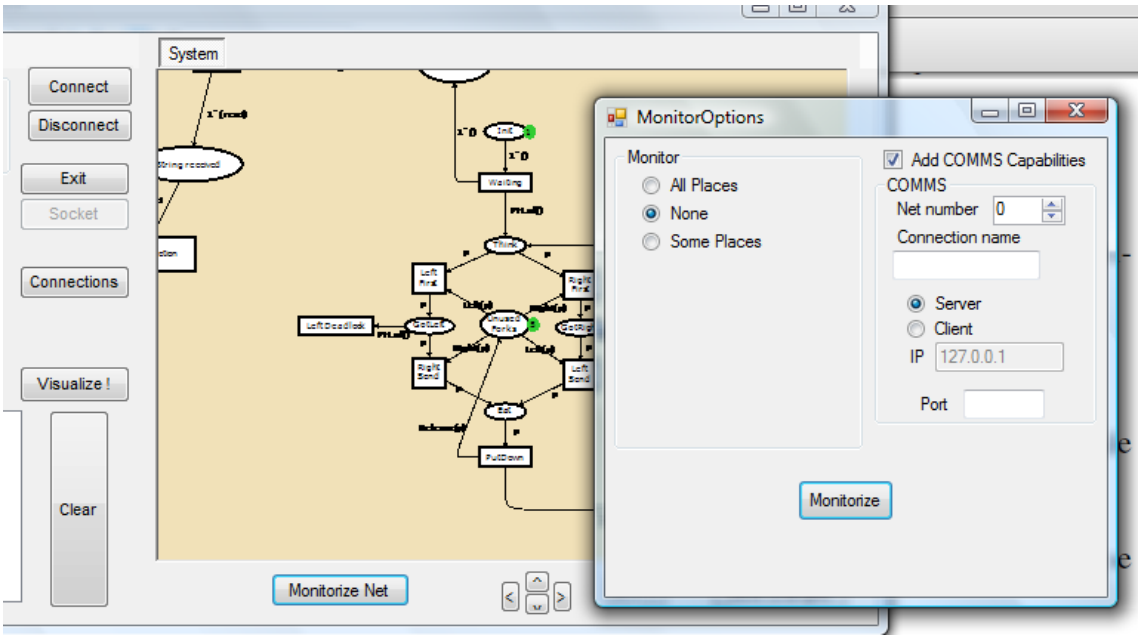


Figura A.15. Seleção dos elementos a monitorar e dos parâmetros da comunicação.

A seleção dos *places* a serem monitorados pode ser feita nesta janela ou diretamente na rede, selecionando um *place* com o botão direito. Porém o monitoramento parcial de alguns *places* não tem muita utilidade normalmente, por isso se recomenda selecionar todos os *places* no caso de querer realizar um monitoramento.

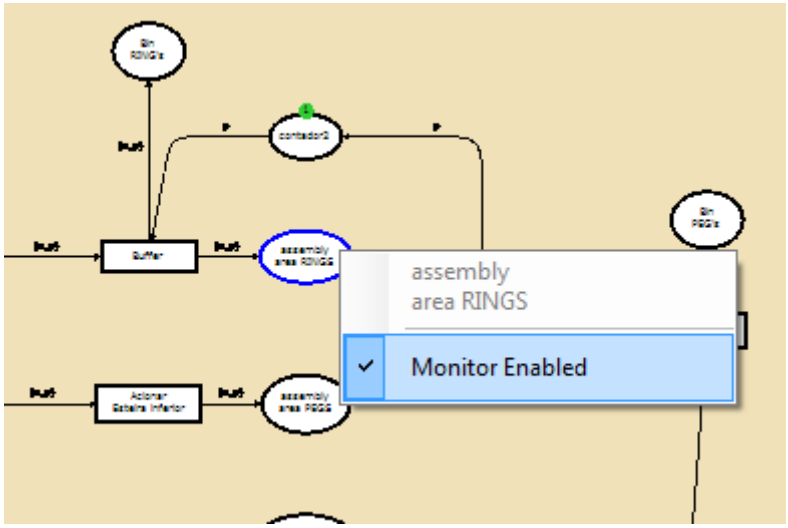


Figura A.16 Seleção de um *place* a monitorar.

V. Instruções de comunicação na CPN

Como mencionado anteriormente, a aplicação desenvolvida permite introduzir instruções de comunicação na CPN sem precisar amplos conhecimentos em SML. Ainda mais, o fato de introduzir estas instruções na aplicação assegura uma maior seguridade para não introduzir erros, melhorando assim a eficiência e a comodidade do usuário.

Lembre-se que unicamente as transições podem efetuar instruções de comunicação. Para adicionar uma instrução, basta selecionar uma transição com o botão direito e selecionar "See instructions".

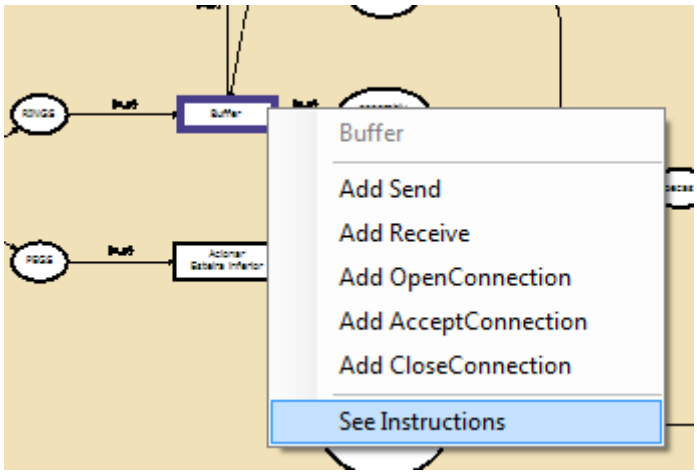


Figura A.17 Seleção de uma transição para incluir uma instrução.

Clicando em qualquer outra instrução, adiciona-se a instrução selecionada de maneira mais rápida seguindo o mesmo procedimento. Sendo este um manual, se mostra um exemplo passo a passo e para usuários principiantes.

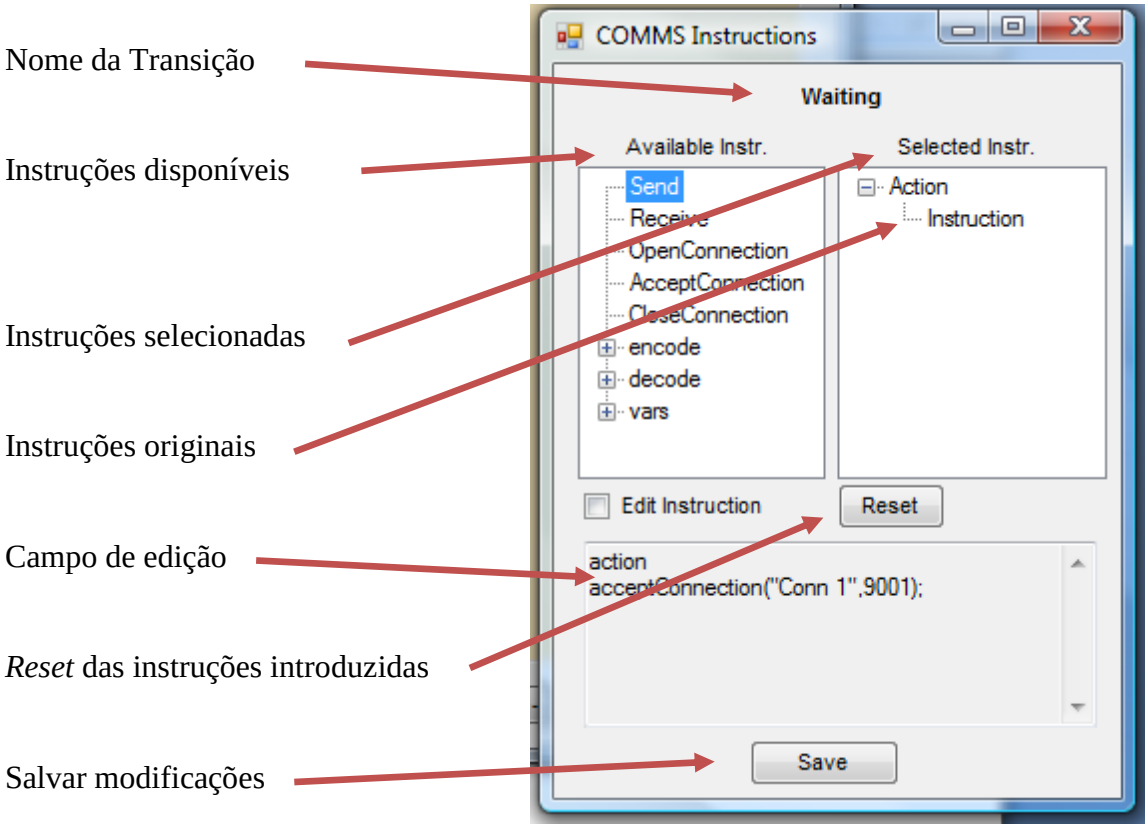


Figura A.18 Aspecto da janela de instruções de comunicações

Unicamente existem 5 instruções disponíveis. As únicas que permitem a customização são a de *Send* e a de *Receive*. Ver declaração destas funções na seção 3.2.3. A função *Send* precisa de dois parâmetros, que são a variável a enviar e a função de codificação. A função *Receive* precisa da função de decodificação unicamente. Para adicionar uma função, basta arrastá-la da esquerda para a direita.

Após adicionar a função *Send* ou *Receive* a aplicação detecta que falta determinar dois parâmetros ou um respectivamente, ressaltando eles em laranja ou vermelho.

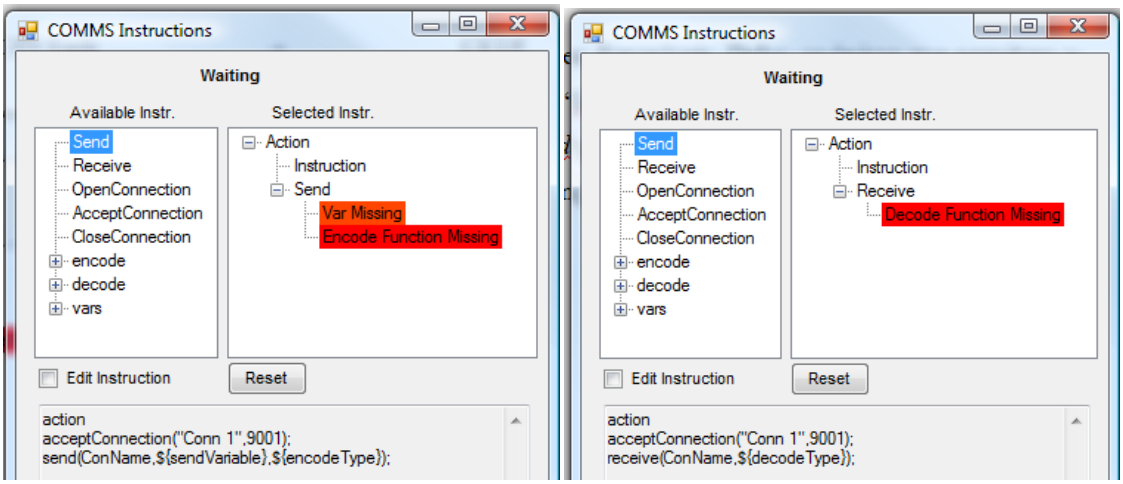


Figura A.19 Inclusão de uma instrução.

Atendendo à mensagem ressaltada, falta adicionar uma variável e uma função de codificação no caso de *Send* e de uma função de decodificação. Simplesmente procede-se arrastando ela desde o grupo correspondente: *encode*, *decode* ou *vars*.

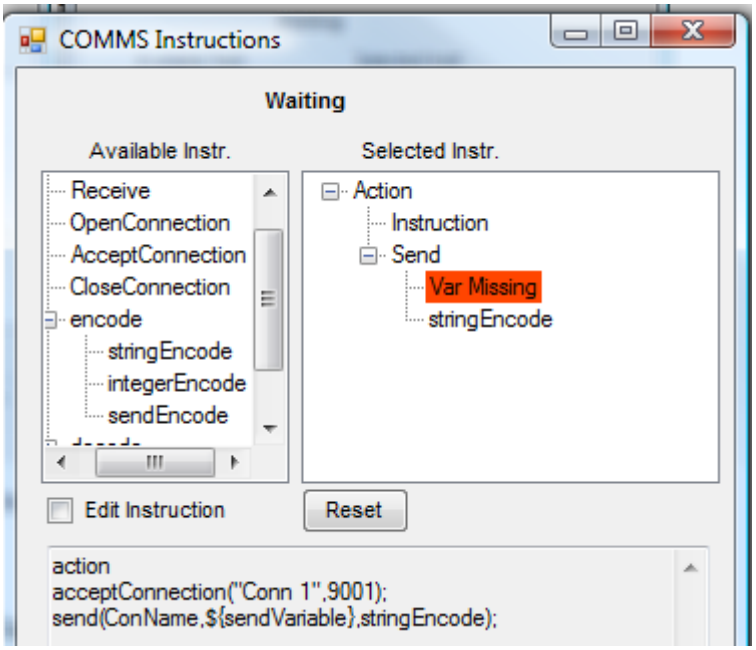


Figura A.20 Inclusão de um parâmetro na instrução *Send*.

Ver que no campo de texto abaixo, foi alterado automaticamente. Outra opção consiste em alterar este campo diretamente mediante a opção “Edit Instruction”.

Uma vez introduzidas as instruções pertinentes clicar em “Save” e a janela será fechada.

VI. Salvar as modificações da CPN

Após adicionar a biblioteca de comunicações, selecionar a opção de monitorar e introduzir instruções de comunicações. Faltaria por incluir estas no arquivo semelhante a XML (cpn). Para isso, clicar no botão “Save Changes”

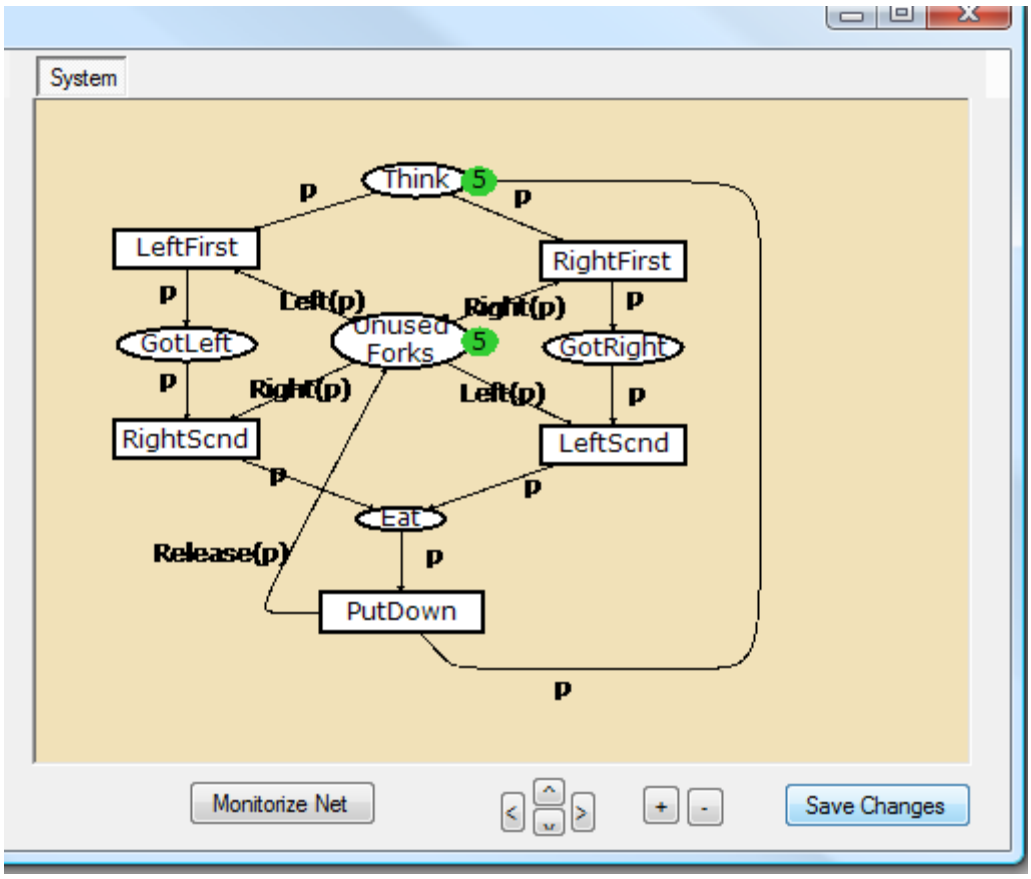


Figura A.21 Salvar alterações na CPN

Recomenda-se veemente que após salvar com as alterações desejadas, a CPN será aberta utilizando CPN Tools e será salva neste programa, pois a CPN Tools checa com maior robustez as possíveis falhas.

VII. Tabela da rede para as provas.

Archivo	Num Rede	Porta	Protocolo
ReceiveProtocolClient.cpn	4	9009	Enabled e Debug
ReceiveProtocolServer.cpn	124	9001	Enabled e Debug
SendProtocolServer.cpn	140	9009	Enabled e Debug
SendProtocolClient.cpn	140	9009	Enabled e Debug
SendProtocolClient2.cpn	141	9010	Enabled e Debug
MonitorTestClient.cpn	140	9009	Enabled e Debug
MonitorTestServer.cpn	140	9009	Enabled e Debug

Tabela A.2 Arquivos para teste das comunicações

VIII. Procedimentos incorretos previstos.

- Conectar duas redes com um mesmo número de rede. Aparecerá uma mensagem de notificação e a segunda conexão não será aprovada.
- Receber informações para enviar a uma rede que não esteja conectada. Aparecerá uma mensagem de notificação e o pacote poderá ser enviado manualmente, embora o protocolo "Enabled (automático)" esteja ativado. A caixa de saída será esvaziada se a rede que enviou a informação for desconectada, perdendo assim essa informação.
- Receber informações para monitorar a CPN sem ter sido visualizada a rede. Aparecerá uma mensagem de notificação.
- Receber informações sob protocolo diferente ao esperado. Aparecerá uma mensagem de notificação.

IX. Conectar a Interface com o exterior

Toda transmissão de dados com a interface, deve ser da seguinte forma:

- Os dados serão segmentados em blocos de no máximo 128 *bytes* antes de ser enviados.
- Desses 128 *bytes*, o primeiro deve conter informações do tamanho do bloco e saber se mais blocos esperam para serem transmitidos. Ver maiores detalhes na Figura 3.2

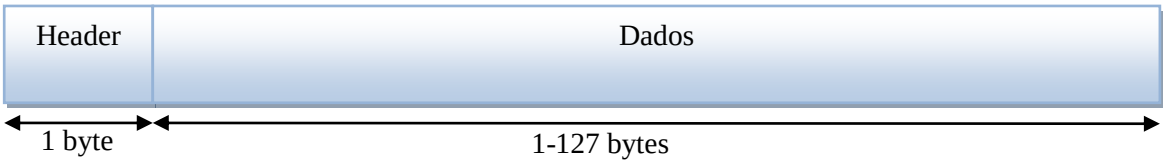


Figura A.22 Distribuição OSI dos blocos de dados para se comunicarem com a Interface

Se o objetivo é se conectar a dispositivos externos com a interface, utilizando o protocolo desenvolvido, é preciso seguir as seguintes indicações:

- Os dados serão também segmentados em blocos de 128 bytes antes de serem enviados. Porém, serão reservados mais 2 ou 3 bytes como cabeçalhos. Se os blocos de dados são maiores que 124-125 bytes, o primeiro terá a estrutura representada na Figura A.23 e os seguintes apresentados na Figura A.22. Ver detalhes de cada cabeçalho na seção 3.1.

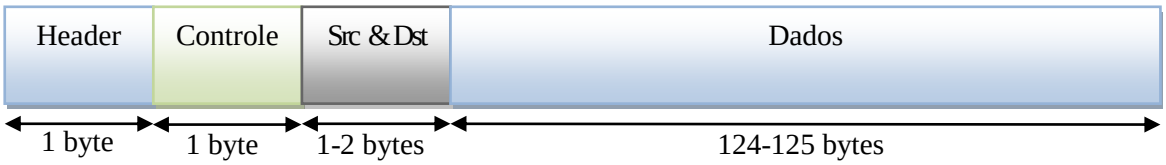


Figura A.23 Distribuição OSI para se comunicar com a Interface utilizando o protocolo desenvolvido

- Em cada transmissão, deve-se tomar em consideração se o formato do cabeçalho *Src&Dst* seja estendido ou não, e habilitar o bit correspondente no byte *Controle.*
- Após a conexão se estabelecer, deve-se enviar informações com o número identificador do módulo conectado e uma *string* menor de 124 bytes com o

nome do dispositivo da seguinte forma: (O bit representando com uma "E" determina se o cabeçalho *Src&Dst* terá 1 ou 2 *bytes*).

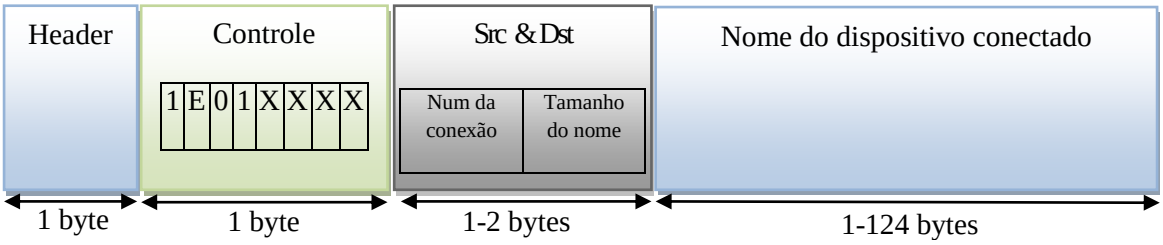


Figura A.24 Distribuição OSI do bloco de dados após a conexão

Se o número da conexão for menor ou igual a 15 ou se o tamanho do nome for menor a 15 caráteres, será utilizado o formato não estendido (E será 0). Se alguma das duas condições anteriores não se efetuar, se ativará o formato estendido (E será 1), permitindo assim que o número da conexão seja de no máximo 255, e o tamanho do nome da conexão de 124.

X. Construir uma função de (de)codificação

Procede-se a explicar passo a passo como construir uma função de codificação ou decodificação na aplicação para utilizá-la depois junto com as instruções de *Send* ou *Receive* segundo o caso. A interface cria um arquivo XML durante sua instalação chamado SMLFunctions.xml. Este arquivo contém a biblioteca desenvolvida neste projeto e é lido cada vez que uma CPN é visualizada pela aplicação. Portanto, se novas funções são adicionadas neste arquivo, serão incorporadas na aplicação. Ver abaixo (XI.Arquivo SMLFunctions.xml) o arquivo simplificado.

Primeiramente vai-se definir o tipo de função. A função a explicar será de decodificação e vai permitir obter os dados sem cabeçalhos utilizando uma transação com o protocolo desenvolvido. Assim sendo, a função a desenvolver é uma mistura entre a *receiveDecode* e a *stringDecode.* Como parâmetros de entrada aceita os dados com os cabeçalhos do protocolo, mas na saída unicamente devolverá os dados sem cabeçalhos. Basicamente, será igual à receiveDecode mas sem decodificar o cabeçalho *Src&Dst*. Chamaremos à função de *receiveSimpleDecode*

Simplesmente, apagam-se as linhas de código não necessárias da função *receiveDecode* como se mostra a seguir:

```
fun receiveDecode(received) =
let
    val Control=addZeros(check(bin(Word8.toInt(Word8Vector.sub(received,0)))))
    val SecByte = Word8.toInt(Word8Vector.sub(received,1))
    val SrcDst = addZeros(check(bin(SecByte)))
    val extended = List.nth(Control,6)
    val Src = if extended=1 then SecByte else bit2Int(List.drop(SrcDst,4),3)
    val Dst = if extended=1 then Word8.toInt(Word8Vector.sub(received,2)) else
      bit2Int(SrcDst,3)
    val Data = Byte.bytesToString(Word8Vector.tabulate (
        Word8Vector.length ( received) - 2-extended, fn n => Word8Vector.sub
       (received, n+2+extended)))
in
     (Src,Dst,Data)
end
```

Figura A.25 Função receiveDecode simplificada

O resultado será o seguinte:

```
fun receiveSimpleDecode(received) =
let
    val Control=addZeros(check(bin(Word8.toInt(Word8Vector.sub(received,0)))))
    val extended = List.nth(Control,6)
    val Data = Byte.bytesToString(Word8Vector.tabulate (
        Word8Vector.length ( received) - 2-extended, fn n => Word8Vector.sub
       (received, n+2+extended)))
in
     (Data)
end
```

Figura A.26 Função receiveSimpleDecode

No último passo, tem que se incorporar esta função no arquivo SMLFunctions.xml. Devido a esta função ser de decodificação será colocada no *tag* chamado "decode" e se criará um *tag* novo com o nome da função nova. Além disso, tem que se introduzir mais um *tag* chamado "ml". Ver abaixo para maiores esclarecimentos.

```
<decode>
  <stringDecode />
  <integerDecode />
  <receiveDecode>
     ......
  </receiveDecode>
  <receiveSimpleDecode>
    <ml id="">
      fun receiveSimpleDecode(received) =
        let
  val Control=addZeros(check(bin(Word8.toInt(Word8Vector.sub(received,0)))))
  val extended = List.nth(Control,6)
  val Data = Byte.bytesToString(Word8Vector.tabulate (
      Word8Vector.length ( received) - 2-extended, fn n => Word8Vector.sub
        (received, n+2+extended)))
      in
        (Data)
      end
    </ml>
  </ receiveSimpleDecode>
</decode>
```

Comprove-se que aparece como opção a escolher dentro da Interface.

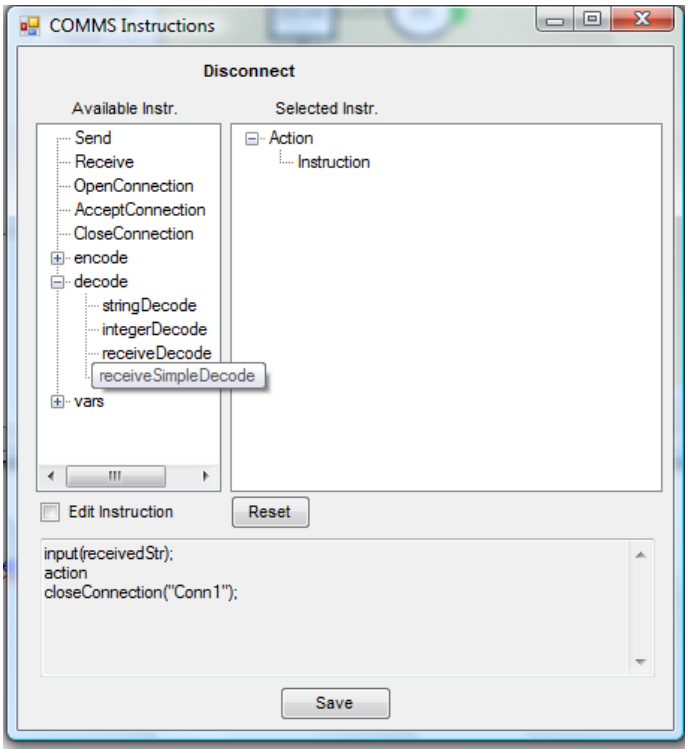


Figura A.27 Função receiveSimpleDecode incorporada

XI. Arquivo SMLFunctions.xml

```
<functions>
  <block id="">
    <id>COMMS</id>
    <globref id="">
      <id>NetNumber</id>
      <ml></ml>
    </globref>
    <ml id="">val ConName=""</ml>
    <ml id="">val Port=0</ml>
    <ml id="">val IP=""</ml>
    <color id="">
      <id>ReceivedTuple</id>
      <product>
        <id>INT</id>
        <id>INT</id>
        <id>STRING</id>
      </product>
    </color>
    <color id="">
      <id>SendData</id>
      <product>
        <id>STRING</id>
        <id>INT</id>
      </product>
    </color>
    <block id="">
      <id>Extra functions</id>
      <ml id="">fun bin(n) = if n=0 then [0] else [n mod 2]^^ bin(n div
2)</ml>
      <ml id="">fun check(n) = if List.last(n) = 0 andalso length(n)&gt;1 then
List.take(n,length(n)-1) else n;</ml>
      <ml id="">fun addZeros(n) = if length(n)&lt;8 then addZeros(n^^[0]) else
n;</ml>
      <ml id="">fun pow(_,0) = 1 | pow(x,n) = x*pow(x,n-1);</ml>
      <ml id="">fun bit2Int(l,0) = List.nth(l,0) | bit2Int(l,n) =
List.nth(l,n)*pow(2,n) + bit2Int(l,n-1);</ml>
    </block>
  </block>
  <ConnInstructions>
    <Send>send(ConName,${sendVariable},${encodeType});</Send>
    <Receive>receive(ConName,${decodeType});</Receive>
    <OpenConnection>openConnection(ConName,IP,Port)</OpenConnection>
    <AcceptConnection>acceptConnection(ConName,Port);</AcceptConnection>
    <CloseConnection>closeConnection(ConName);</CloseConnection>
  </ConnInstructions>
  <encode>
    <stringEncode/>
    <integerEncode/>
    <sendEncode>
      ......
    </sendEncode>
  </encode>
  <decode>
    <stringDecode />
    <integerDecode />
    <receiveDecode>
      ......
    </receiveDecode>
  </decode>
  <connection>
    <connectEncode>
      ......
    </connectEncode>
    <connectClient>
```

```
      ......
    </connectClient>
    <connectServer>
      ......
    </connectServer>
  </connection>
  <monitor>
    <COMMSMonitor>
      ......
    </COMMSMonitor>
    <monitorEncode>
      ......
    </monitorEncode>
  </monitor>
  <extra>
    <INT>
      <color id="">
        <id>INT</id>
        <int/>
      </color>
    </INT>
    <STRING>
      <color id="">
        <id>STRING</id>
        <string/>
      </color>
    </STRING>
    <connected>
      <globref id="">
        <id>connected</id>
        <ml>false</ml>
        <layout>globref connected = false</layout>
      </globref>
    </connected>
  </extra>
</functions>
```